Gestión de Calidad del software

En este apartado se van a presentador los conceptos básicos y la terminología propia de esta área.

En primer lugar aclararemos qué se entiende por calidad en el contexto de la ingeniería del software.

A continuación pasaremos a analizar qué estructura y utilidad tiene un modelo de calidad.

1.1. ¿Qué es la Calidad del Software?

La calidad está de moda, en todos los aspectos, pero especialmente en el desarrollo de software. El interés por la calidad crece de forma continua, a medida que los clientes se vuelven más selectivos y comienzan a rechazar los productos poco fiables o que realmente no dan respuesta a sus necesidades. Ahora bien, ¿qué es la calidad del software?

A la hora de definir la calidad del software se pueden adoptar diferentes enfoques.

Como primer paso es importante diferenciar entre la calidad del producto software y la calidad del proceso de desarrollo. No obstante, las metas que se establezcan para la calidad del producto van a determinar las metas a establecer para la calidad del proceso de desarrollo, ya que la calidad del producto va a estar en función de la calidad del proceso de desarrollo. Sin un buen proceso de desarrollo es casi imposible obtener un buen producto.

La calidad del producto software se diferencia de la calidad de otros productos de fabricación industrial, ya que el software tiene ciertas características especiales:

- El software es un producto mental; no restringido por las leyes de la Física o por los límites de los procesos de fabricación. Es algo abstracto, y su calidad también lo es.
- Se desarrolla, no se fabrica. El coste está fundamentalmente en el proceso de diseño, no en la producción. Y los errores se introducen también en el diseño, no en la producción.
- El software no se deteriora con el tiempo. No es susceptible a los defectos del entorno, y su curva de fallos es muy diferente de la del hardware. Todos los problemas que surjan durante el mantenimiento estaban allí desde el principio, y afectan a todas las copias del mismo; no se generan nuevos errores.
- Es artesanal en gran medida. El software, en su mayoría, se construye a medida, en vez de ser construido ensamblando componentes existentes y ya probados, lo que dificulta aún más el control de su calidad. Aunque se ha escrito mucho sobre la reutilización del software, hasta ahora se han conseguido pocos éxitos tangibles.
- El mantenimiento del software es mucho más complejo que el mantenimiento del hardware. Cuando un componente hardware se deteriora se sustituye por una pieza de repuesto, pero cada fallo en el software implica un error en el diseño o en el proceso mediante el cual se tradujo el diseño en código máquina ejecutable.
- Es engañosamente fácil realizar cambios sobre un producto software, pero los efectos de estos cambios se pueden propagar de forma explosiva e incontrolada.
- Como disciplina, el desarrollo de software es aún muy joven, por lo que las técnicas de las que disponemos aún no son totalmente efectivas o no están totalmente calibradas.

- El software con errores no se rechaza. Se asume que es inevitable que el software presente errores.

También es importante destacar que la calidad de un producto software debe ser considerada en todos sus estados de evolución (especificaciones, diseño, código...). No basta con tener en cuenta la calidad del producto una vez finalizado, cuando los problemas de mala calidad ya no tienen solución o la solución es muy costosa.

Los principales problemas a los que nos enfrentamos a la hora de hablar de la calidad de un producto software son:

- La definición misma de la calidad del software: ¿es realmente posible encontrar un conjunto de propiedades en un producto software que nos den una indicación de su calidad? Para dar respuesta a estas preguntas aparecen los modelos, de calidad.
  - Modelos de calidad: en los modelos de calidad, la calidad se define de forma jerárquica. Resuelven la complejidad mediante la descomposición. La calidad es un concepto que se deriva de un conjunto de sub-conceptos.
- La comprobación de la calidad del software ¿cómo medir el grado de calidad de un producto software? Aquí aparece el concepto de control de calidad, del que hablaremos en el apartado 2.
  - Control de calidad: actividades para evaluar la calidad de los productos desarrollados.
- La mejora de la calidad del software: ¿cómo utilizar la información disponible acerca de la calidad del producto software para mejorar su calidad a lo largo del ciclo de vida? Tal y como veremos, no sólo es posible «medir» la calidad, sino también «construir» la calidad durante el proceso de desarrollo del producto. En este eje aparecen dos conceptos importantes:
  - Gestión de calidad: determinación y aplicación de las políticas de calidad de la empresa (objetivos y directrices generales).
  - Garantía o aseguramiento de calidad: conjunto de actividades planificadas y sistemáticas necesarias para proporcionar confianza en que el producto software satisfará los requisitos dados de calidad.

En este apartado nos vamos a ocupar de la primera cuestión, la definición de la calidad.

Veamos la definición que dan de la calidad algunos de los padres de la idea de la gestión de la calidad:

Definiciones:

Según *Deming*, calidad es "conformidad con los requisitos y confianza en el funcionamiento."

Según *Jurán*, calidad es "adecuación para su uso"

Crosby pone más énfasis en la prevención: "hacerlo bien a la primera"

Veamos otras definiciones de calidad extraídas de estándares internacionales:

> **Definiciones:**
>
> "La calidad es la suma de todos aquellos aspectos o características de un producto o servicio que influyen en su capacidad para satisfacer las necesidades, expresadas o implícitas" (ISO 8402)
>
> "Grado con el cual el cliente o usuario percibe que el software satisface sus expectativas" (IEEE 729-83)
>
> "Capacidad del producto software para satisfacer los requisitos establecidos" (DoD 2168)

Lo que está claro a partir de estas definición es que la calidad es algo relativo. Siempre va a depender de los requisitos o necesidades que se desee satisfacer. Por eso la evaluación de la calidad de un producto siempre va a implicar una comparación requisitos preestablecidos y el producto realmente desarrollado.

El problema es que, por lo general, una parte de los requisitos van a estar explícitos (se encontrarán en la especificación de requisitos de software, tanto los funcionales como otros requisitos), pero otra parte van a quedar implícitos (el usuario sabe lo que quiere pero no siempre es capaz de expresarlo). Hay que intentar que queden implícitos la menor cantidad de requisitos posible. No se podrá conseguir un producto de buena calidad sin una buena especificación de requisitos de software.

Teniendo esto en cuenta, en un producto software vamos a tener diferentes visiones de la calidad:

- Necesaria o requerida: la que quiere el cliente.
- Programada o especificada: la que se ha especificado explícitamente y se intenta conseguir.
- Realizada: la que se ha conseguido.

Nuestro objetivo es conseguir que las tres visiones coincidan. A la intersección entre la calidad requerida y la calidad realizada se le llama calidad percibida, y es la única que el cliente valora. Toda aquella calidad que se realiza pero no se necesita es un gasto inútil de tiempo y dinero.

También está claro que las definiciones que se han dado de la calidad son demasiado generales e imprecisas como para que resulten de utilidad a la hora de construir un software de calidad. Por eso surge el concepto de modelo de calidad, que nos ayuda a definir la calidad del software de una forma más precisa y útil.

1.2. Modelos de Calidad del Software

1.2.1. Estructura de los modelos de calidad

Los modelos de calidad del software vienen a ayudar en la puesta en práctica del concepto general de calidad que vimos en el apartado anterior, ofreciendo una definición operacional.

Uno de los modelos de calidad más antiguo y extendido es el de McCall [McCall, 1977], y de él han derivado otros modelos, como el de Boehm [Boehm,1978] o el SQM [Murine, 1988].

En los modelos de calidad, la calidad se define de forma jerárquica. Es un concpto que se deriva de un conjunto de subconceptos, cada uno los cuales se va a evaluar a través de un conjunto de indicadores o métricas.

Tienen una estructura, por lo general, en tres niveles:

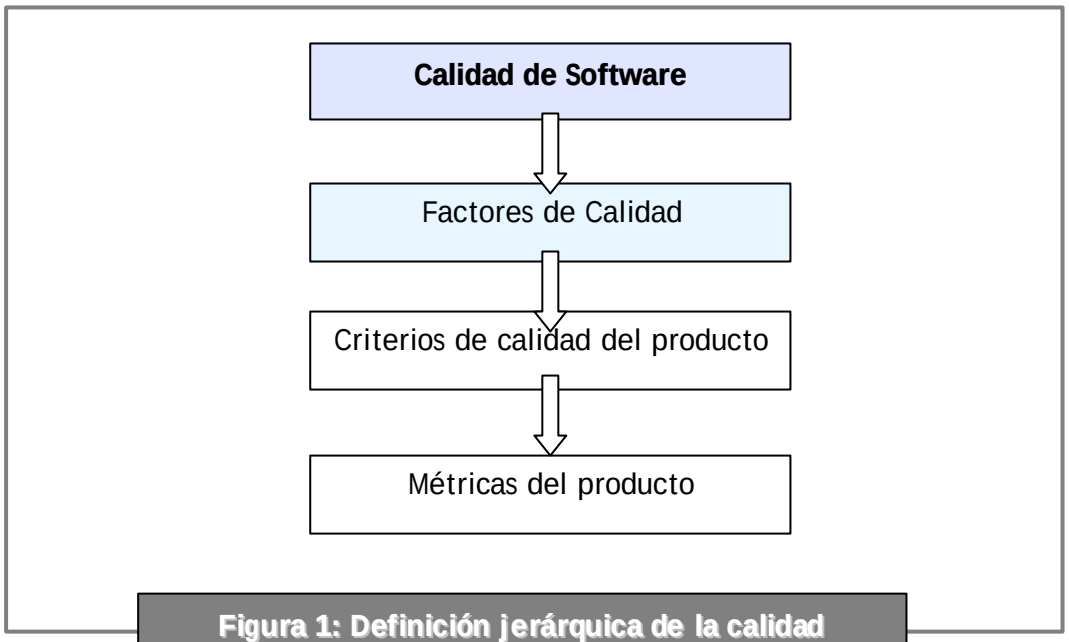


Figura 1: Definición jerárquica de la calidad

- En el nivel más alto de la jerarquía se encuentran los **factores** de calidad, que representan la calidad desde el punto de vista del usuario. Son las características que componen la calidad. También se les llama atributos de calidad externos.
- Cada uno de los factores se descompone en un conjunto de **criterios** de calidad. Son atributos que, cuando están presentes, contribuyen al aspecto de la calidad, que el factor asociado representa. Se trata de una visión de la calidad desde el punto de vista del producto software. También se les llama atributos de calidad internos.
- Para cada uno de los criterios de calidad se definen entonces un conjunto de **métricas**, que son medidas cuantitativas de ciertas características del producto que, cuando están presentes, dan una indicación del grado en que dicho producto posee un determinado atributo de calidad.

La ventaja de los modelos de calidad es que la calidad se convierte en algo concreto, que se puede definir, que se puede medir y, sobre todo, que se puede planificar.

Los modelos de calidad, ayudan también a comprender las relaciones que existen entre diferentes características de un producto software.

Una desventaja es que aún no ha sido demostrada la validez absoluta de ninguno de estos modelos. Las conexiones que se establecen entre características, atributos y métricas se derivan de la experiencia, de ahí que existan múltiples modelos.

1.2.2. El modelo de MeCall

Como ejemplo vamos a ver el modelo de McCall.

El modelo de McCall organiza los factores en tres ejes o puntos de vista desde los cuales el usuario puede contemplar la calidad de un producto:

- ✓ Operación del producto.

- ✓ Revisión del producto.
- ✓ Transición del producto.

El modelo de McCall se basa en 11 factores de calidad, que se organizan en torno a los tres ejes de la siguiente forma:

PUNTO DE VISTA	FACTORES
OPERACIÓN DEL PRODUCTO	- Facilidad de uso - Integridad - Corrección - Fiabilidad - Eficiencia
REVISIÓN DEL PRODUCTO	- Facilidad de mantenimiento - Facilidad de prueba - Flexibilidad
TRANSICIÓN DEL PRODUCTO	- Facilidad de reutilización - Interoperabilidad - Portabilidad

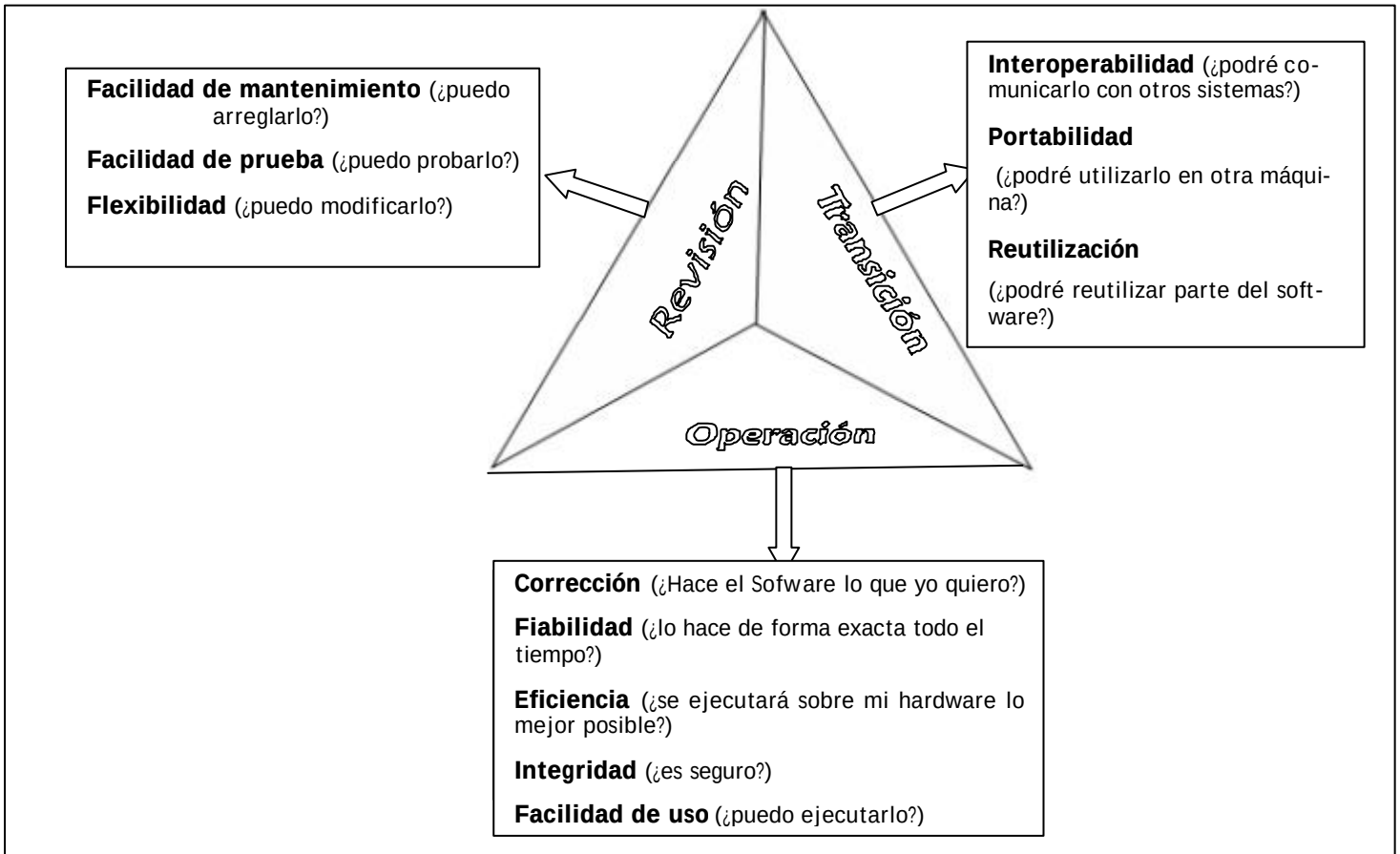


Figura 2 **Factores de calidad según Mac Call**

Los factores de McCall se definen como sigue:

1. **Corrección**. Hasta qué punto un programa cumple sus especificaciones y satisface los objetivos del usuario. Por ejemplo, si un programa debe ser capaz de sumar dos números y en lugar de sumar los multiplica, es un programa incorrecto. Es quizás el factor más importante, aunque puede no servir de nada sin los demás factores.
2. **Fiabilidad**. Hasta qué punto se puede confiar en el funcionamiento sin errores del programa. Por ejemplo, si el programa anterior suma dos números, pero en un 25 por ciento de los casos el resultado que da no es correcto, es poco fiable.
3. **Eficiencia**. Cantidad de código y de recursos informáticos (CPU, memoria) que precisa un programa para desempeñar su función. Un programa que suma dos números y necesita 2 MB de memoria para funcionar, o que tarda dos horas en dar una respuesta, es poco eficiente.
4. **Integridad**. Hasta qué punto se controlan los accesos ilegales a programas o datos. Un programa que permite el acceso de personas no autorizadas a ciertos datos es poco íntegro.
5. **Facilidad de uso**. El coste y esfuerzo de aprender a manejar un producto, preparar la entrada de datos e interpretar la salida del mismo.
6. **Facilidad de mantenimiento**. El coste de localizar y corregir defectos en un programa que aparecen durante su funcionamiento.
7. **Facilidad de prueba**. El coste de probar un programa para comprobar que satisface sus requisitos. Por ejemplo, si un programa requiere desarrollar una simulación completa de un sistema para poder probar que funciona bien, es un programa difícil de probar.
8. **Flexibilidad**. El coste de modificación del producto cuando cambian sus especificaciones.
9. **Portabilidad (o transportabilidad)**. El coste de transportar o migrar un producto de una configuración hardware o entorno operativo a otro.
10. **Facilidad de reutilización**. Hasta qué punto se puede transferir un módulo o programa del presente sistema a otra aplicación, y con qué esfuerzo.
11. **Interoperabilidad**. El coste y esfuerzo necesario para hacer que el software pueda operar conjuntamente con otros sistemas o aplicaciones software externos.

Cada uno de estos factores se descompone, a su vez, en criterios. En el modelo de McCall se definen un total de 23 criterios, con el siguiente significado:

1. **Facilidad de operación**. Atributos del software que determinan la facilidad de operación del software.
2. **Facilidad de comunicación**. Atributos del software que proporcionan al usuario entradas y salidas fácilmente asimilables.
3. **Facilidad de aprendizaje**. Atributos del software que facilitan la familiarización inicial del usuario con el software y la transición desde el modo actual de operación.
4. **Control de accesos**. Atributos del software que proporcionan control de acceso al software y los datos que maneja.
5. **Facilidad de auditoría**. Atributos del software que facilitan el registro y la auditoría de los accesos al software.

6. **Eficiencia en ejecución.** Atributos del software que minimizan el tiempo de procesamiento.
7. **Eficiencia en almacenamiento**. Atributos del software que minimizan el espacio de almacenamiento necesario.
8. **Precisión**. Atributos del Software que proporcionan el grado de precisión requerido en los cálculos y los resultados.
9. **Consistencia**. Atributos del software que proporcionan uniformidad en las técnicas y notaciones de diseño e implementación utilizadas.
10. **Tolerancia a fallos**. Atributos del software que posibilitan la continuidad del funcionamiento bajo condiciones no usuales.
11. **Modularidad**. Atributos del software que proporcionan una estructura de módulos altamente independientes.
12. **Simplicidad**. Atributos del software que posibilitan la implementación de funciones de la forma más comprensible posible.
13. **Completitud**. Atributos del software que proporcionan la implementación completa de todas las funciones requeridas.
14. **Trazabilidad (rastreabilidad)**. Atributos del software que proporcionan una traza desde los requisitos a la implementación con respecto a un entorno operativo concreto.
15. **Auto descripción**. Atributos del software que proporcionan explicaciones sobre la implementación de las funciones.
16. **Capacidad de expansión**. Atributos del software que posibilitan la expansión del software en cuanto a capacidades funcionales y datos.
17. **Generalidad**. Atributos del software que proporcionan amplitud a las funciones implementadas.
18. **Instrumentación**. Atributos del software que posibilitan la observación del comportamiento del software durante su ejecución (para facilitar las mediciones del uso o la identificación de errores).
19. **Independencia entre sistema y software**. Atributos del software que determinan su independencia del entorno operativo.
20. **Independencia del hardware**. Atributos del software que determinan su independencia del hardware.
21. **Compatibilidad de comunicaciones**. Atributos del software que posibilitan el uso de protocolos de comunicación e interfaces estándar.
22. **Compatibilidad de datos**. Atributos del software que posibilitan el uso y representaciones de datos estándar.
23. **Concisión**. Atributos del software que posibilitan la implementación de una función con la menor cantidad de código posible.

La relación factores-criterios que establece el modelo queda plasmada en la siguiente tabla:

Factor	Criterio
Facilidad de uso	Facilidad de operación Facilidad de comunicación Facilidad de aprendizaje
Integridad	Control de accesos Facilidad de auditoría
Corrección	Completitud Consistencia Trazabilidad
Fiabilidad	Precisión Consistencia Tolerancia a fallos Modularidad Simplicidad
Eficiencia	Eficiencia en ejecución Eficiencia en almacenamiento
Facilidad de mantenimiento	Modularidad Simplicidad Consistencia Cocisión Autodescripción
Facilidad de prueba	Modularidad Simplicidad Autodescripción Instrumentación
Flexibilidad	Autodescripción Capacidad de expansión Generalidad Modularidad
Reusabilidad	Autodescripción Generalidad Modularidad Independencia entre sistema y software Independencia del hardware
Interoperabilidad	Modularidad Compatibilidad de comunicaciones Compatibilidad de datos
Portabilidad	Autodescripción Modularidad

Factor	Criterio
Portabilidad	Independencia entre sistema y software Independencia del hardware.

En cuanto a las métricas, McCall propuso 41 métricas, sobre todo métricas de tipo subjetivo, es decir, métricas que evaluadas por personas diferentes podrían dar como resultado valores diferentes. Aún hoy en día, no hay métricas formales y objetivas que cubran todos los criterios del modelo de McCall.

Vamos a ver como ejemplo las métricas asociadas al criterio de calidad «completitud», dentro del factor de calidad «corrección», según McCall. '

Para evaluar la completitud es necesario dar respuesta a la siguiente lista de comprobación:

1. No hay referencias ambiguas (R, D, I).
2. Todas las referencias a datos definidas son calculadas u obtenidas de una fuente externa (R, D, I).
3. Todas las funciones definidas son utilizadas (R, D, I).
4. Todas las referencias a funciones están definidas (R, D, I).
5. Se han definido, todas las condiciones y procesamientos para cada punto de decisión (R, D, I).
6. Concuerdan todos los parámetros de llamada a funciones definidos y referenciados (D, I).
7. Todos los informes de problemas se han resuelto (R, D, I).
8. El diseño concuerda con los requisitos (D).
9. El código concuerda con el diseño (I).

Las letras R, D e I indican si la lista de comprobación es aplicable a los requisitos (R), el diseño (D) y/o la implementación (I).

Se contesta a estas pregunta con un Sí o No, y luego se aplica la siguiente fórmula matemática que da como resultado el grado o nivel de calidad para dicho atributo:

$$\frac{\text{(Número de Sí para R/6) + (Número de Sí para D/8) + Número de Sí para I/8)}}{3}$$

De esta forma, la medida para la completitud es un número entre 0 y 1.

En general, el modelo de McCall propone asociar a cada criterio una fórmula de regresión del tipo:

$$CC = r_1 m_1 + r_2 m_2 + \ldots + r_n m_n$$

Donde los r_j son los coeficientes de regresión, y donde los m_j representan las distintas métricas asociadas al criterio. De la misma forma se propagarán a los factores los valores calculados para los criterios.

La medida para la corrección, por ejemplo, se calculará aplicando la fórmula:

$$\frac{x + y + z}{3}$$

Donde x es la medida para la completitud, y la medida para la trazabilidad y z la medida para la consistencia.

Para que todas las métricas se puedan combinar sin problemas, lo habitual es que se normalicen sus valores dentro del intervalo [0,1].

1.2.3. Otras métricas

Otros ejemplos de métricas son los siguientes:

Fiabilidad	=	1 - (número de errores/número de Líneas de código)
Facilidad de mantenimiento	=	1 - 0,1 (número medio de días-hombre por corrección)
.../...		
Portabilidad	=	1 – (esfuerzo para portar/esfuerzo para implementar)
Flexibilidad	=	1 – 0,05 (número medio de días-hombre por cambio)

Otros indicadores que se pueden utilizar para evaluar la **fiabilidad** de un producto son los siguientes:

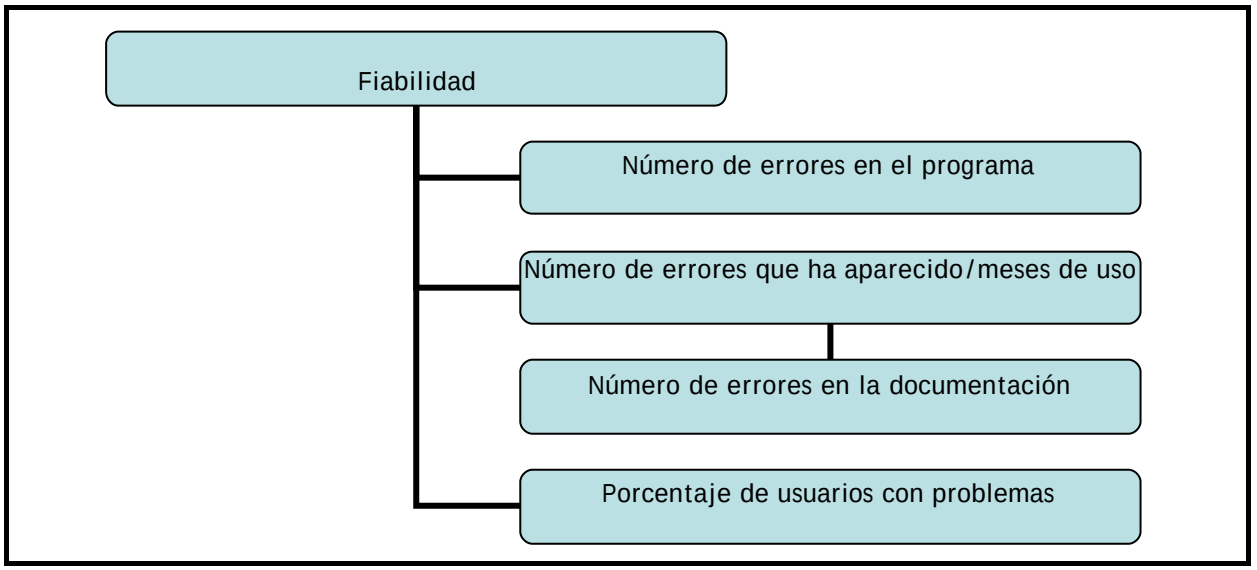


Figura 3 **Indicadores para evaluar la fiabilidad**

La medida de la facilidad de uso se puede basar en la probabilidad de que el operador del sistema no se encuentre con problemas en la interfaz de usuario durante un cierto período de operación. El problema de esta métrica es recoger los datos necesarios para calcular la función de distribución de probabilidad de los problemas con la interfaz.

Los manuales bien estructurados, los mensajes de error informativos, las funciones de ayuda y las interfaces consistentes también contribuyen a la facilidad de uso, pero no basta con contar el número de mensajes o pantallas de ayuda para tener una métrica de la facilidad de uso.

Otras métricas propuestas se refieren a la comprensibilidad o legibilidad de las pantallas de ayuda y los textos de mensajes de error y manuales. Estas métricas son sin embargo poco indicativas.

Otras métricas se refieren al esfuerzo necesario para aprender a manejar el sistema, o a la velocidad a la que trabaja el usuario una vez entrenado, o a los errores que se cometen en

el trabajo normal. Pero para medir lo agradable que resulta la interfaz, lo mejor es hacer encuestas a los usuarios.

En cuanto a la **facilidad de mantenimiento**, la métrica ideal sería la probabilidad de que la causa de un incidente será diagnosticada o un fallo corregido con una cierta cantidad de esfuerzo en un determinado entorno.

Otras métricas se refieren al proceso de mantenimiento, como:

- ✓ Tiempo medio de reparación o cambio.
- ✓ Número de problemas sin resolver.
- ✓ Tiempo empleado en problemas sin resolver.
- ✓ Porcentaje de cambios que introducen defectos.
- ✓ Número de módulos afectados por cada cambio.

También las métricas de complejidad se utilizan como indicadores de la facilidad de mantenimiento. La estructuración del código y de la documentación asociada suele estar muy relacionada con la facilidad de mantenimiento. Pero es más una forma de identificar potenciales puntos peligrosos que una predicción de pobre mantenibilidad.

1.3. ¿Cómo utilizar un Modelo de Calidad?

1.3.1. Estrategias de uso de un modelo de calidad

Dependiendo del grado de conformidad con el modelo de calidad seleccionado como referencia para un proyecto, se pueden adoptar dos estrategias:

- MODELO FIJO:
  - Se aceptan los factores, criterios y métricas que propone el modelo.
  - Se aceptan las relaciones entre factores y criterios, y entre criterios y métricas.
  - Sólo es necesario seleccionar un subconjunto de los factores de calidad como requisitos de calidad para el proyecto.
- DEFINICIÓN PARTICULAR DE LA CALIDAD:
  - Se acepta la filosofía de la descomposición.
  - Se selecciona un subconjunto de los factores de calidad como requisitos de calidad para el proyecto.
  - Se decide la descomposición más adecuada para los factores de calidad seleccionados.

1.3.2. Pasos para el uso de un modelo de Calidad

1.3.2.1. Al principio del proyecto

Al especificar la calidad requerida de un producto software hay que:

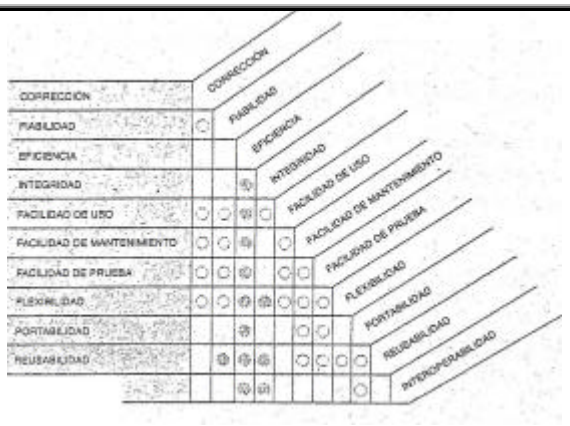
1. Seleccionar cuáles de los factores de calidad van a ser requisitos de calidad del sistema. Para ello hay que tener varias cosas en consideración:
   - La relación que tienen los factores con las características peculiares del producto o proyecto. Así, por ejemplo, si se espera que el ciclo de vida del sistema sea largo la «facilidad de mantenimiento» y la «flexibilidad» se convierten en un requisito; sí el sistema es experimental y se espera que las especificaciones del sistema cambien frecuentemente, la «flexibilidad» será importante y sin embargo la «eficiencia» apenas tendrá importancia; si el sistema se desarrolla para

un entorno en el que el hardware evoluciona rápidamente, la «portabilidad» es esencial; si se espera que ciertas funciones del sistema se utilicen por un largo período de tiempo, aunque el resto del sistema cambie, la «facilidad de reutilización» será fundamental, etc.

- El coste del factor de calidad frente al beneficio que proporciona. La siguiente tabla indica, para cada factor, el ahorro que se puede esperar cuando se consigue frente al coste necesario para conseguir dicho factor.

Factor	Beneficio frente al coste
Corrección	Alto
Fiabilidad	Alto
Eficiencia	Bajo
Integridad	Bajo
Facilidad de uso	Medio
Facilidad de mantenimiento	Alto
Facilidad de prueba	Alto
Flexibilidad	Medio
Portabilidad	Medio
Reusabilidad	Medio
Interoperatibilidad	Bajo

- Las implicaciones de los factores de calidad sobre el ciclo de vida, es decir, en qué etapas es necesario evaluar cada uno. de los factores de calidad, y en qué etapas se dejan sentir los efectos de una calidad pobre con respecto a cada uno de estos factores.
- Las interrelaciones entre factores. Algunos factores pueden ser conflictivos entre sí. La eficiencia, por ejemplo, está en conflicto con prácticamente todos los demás factores de calidad. La siguiente tabla indica la dependencia entre los factores de McCall.



Cuando hay un alto grado de calidad para un factor, ¿qué grado de calidad se puede esperar de otros factores?

? Bajo

º Alto

Tabla 1 **Dependencia entre los factores de McCall**

2. Una vez seleccionados los factores de calidad que son requisitos para el producto, es necesario organizarlos en orden de *importancia*.
3. Una vez establecidos los factores de calidad, el modelo de calidad proporciona automáticamente el conjunto de atributos o *criterios* relacionados con dichos factores.
4. Para cada uno de los criterios de calidad se definen o eligen entonces un conjunto de *métricas*.
5. Se debe entonces establecer *valores deseables* para los criterios en función de datos históricos, el promedio en la industria, etc. Se pueden establecer valores finales, es decir, los que se desea obtener una vez finalizado el desarrollo, y también valores intermedios o predictivos en cada período de medición durante el desarrollo.
6. Por último, se deberán establecer los valores *mínimos* aceptables.

La explicación para cualquier selección o decisión deberá ser adecuadamente documentada.

1.3.2.2. Durante el desarrollo

Todo lo anterior se realizará al principio del proyecto. Ahora bien, durante el desarrollo será necesario:

- Implementar las métricas, es decir, tomar las medidas necesarias.
- Analizar los resultados de las métricas.
- Tomar medidas correctivas si es necesario, es decir, si los valores obtenidos están por debajo de los valores mínimos aceptables. Estas medidas correctivas pueden afectar tanto al proceso de desarrollo como al proceso de gestión.

1.3.2.3. Al final del proyecto

Una vez finalizado el proyecto, será necesario validar las medidas predictivas utilizadas, y comprobar si en efecto se pueden tomar como indicadores de los valores finales.

1.4 La visión simplista de la Calidad

Los modelos de calidad requieren una planificación detallada y una cuidadosa recolección de medidas. Puede ser muy costoso incluso para un número reducido de factores, por lo que requieren recursos extra y, como consecuencia de ello, se usan con poca frecuencia.

En la mayor parte de los casos se adopta una visión de la calidad mucho más restringida, basada únicamente en el número de fallos (defectos conocidos).

La métrica de calidad única que se utiliza es:

DENSIDAD DE DEFECTOS = Número de defectos / Tamaño del producto

Oscila en la industria, según datos publicados, entre 2 y 60 defectos/KNCSS, donde KNCSS significa *Kilo Non Comment Source Statements*, es decir, 1.000 instrucciones de código fuente sin comentarios.

Esta métrica se puede usar no sólo para código, sino también para análisis y diseño, tomando una definición de defecto adecuada, basada en el número de cambios requeridos.

El peligro de esta aproximación es el de utilizar mal esta medida simplificada de la calidad.

En primer lugar, no todos los defectos conducen aun fallo. Sólo los fallos son percibidos por el usuario y son los que, por lo tanto, inciden en la calidad. Según un estudio realizado por Adams, en sistemas grandes y complejos un tercio de los defectos totales conducen a fallos que sólo ocurren como promedio cada 5.000 años de tiempo de ejecución o más, y sólo un 2 por ciento de los defectos son responsables de los fallos que ocurren cada cinco años o menos.

Puede haber productos con un número de defectos alto que sin embargo apenas fallen, y que serán percibidos por el usuario como de alta calidad.

Por otro lado, esta medida está muy relacionada con la forma y el proceso de búsqueda y detección de defectos. Nos puede decir más de la calidad de este proceso que del producto. Un producto en el que se han detectado pocos defectos puede indicar que el proceso de pruebas ha sido poco exhaustivo y la mayor parte de los defectos permanecen ocultos.

1.5. Terminología

Error

Se define como una incorrección cometida por un humando durante el proceso de desarrollo

Defecto

Es la consecuencia de un error. Así, por ejemplo, si una función tiene el objetivo de sumar 10 al calor que recibe como entrada, y en realidad está sumando 20, eso es un defecto, debido al error del programador que escribió 20 en lugar de 10.

Fallo (Failure)

Se entiende por la manifestación de un defecto en el software. Por ejemplo, cuando a la función anterior se le da como entrada el valor 10 y la salida que se obtiene es 30 en lugar de 20, que es el valor esperado.

Fallas (Fault)

Las fallas son los defectos que aún no ha sido detectados y eliminados cuando comienzan las pruebas. Algunas de estas fallas se convertirá en fallos si se detectan durante las pruebas o el uso del sistema.

Incidente

Se llama a una situación en la que se produce y se informa de un comportamiento no esperado en el sistema.

Defecto

Es una desviación en el valor esperado para una cierta característica. Los defectos no tienen por qué afectar al funcionamiento del objeto defectuoso. Un programa poco mantenible, por ejemplo, puede ser totalmente correcto

2. Actividades del control de calidad

El objetivo de las actividades de control de calidad es comprobar si un producto posee o no una determinada característica de calidad en el grado requerido. Cuando un producto no posee una determinada característica de calidad se dice que tiene un **defecto**. Por lo tan-

to, se puede decir también que el objetivo del control de calidad es identificar defectos en el producto y corregirlos.

Se pueden clasificar las actividades de control de calidad en dos categorías: controles estáticos y controles dinámicos. Los primeros analizan el objeto sin necesidad de ejecutarlo mientras que los segundos requieren la ejecución del objeto que está siendo probado.

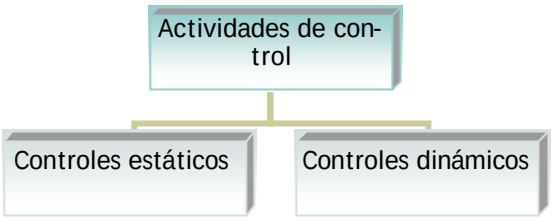


Figura 4 **Actividades del control de calidad**

La barrera entre controles estáticos y dinámicos no es totalmente estricta. Cualquier forma de control dinámico requiere un cierto grado de análisis estático. Además hay algunas técnicas, como la verificación formal y la ejecución simbólica, consideradas como estáticas, que «ejecutan» el código, aunque en un entorno no real.

2.1. Controles Estáticos

Vamos a clasificar las actividades de control estático de la siguiente forma:

Figura 5 **Actividades del control estático**

A continuación vamos a ver en qué consiste cada uno de ellos.

2.1.1. Controles estáticos manuales informales

Estas actividades las realizan los propios autores de los objetos a comprobar, o personas de su misma categoría y ocupación.

Comprobación de escritorio (desk checking):

Consiste en examinar a mano e individualmente el objeto que se acaba de desarrollar. Es el método más tradicional para analizar un programa.

Se debe aplicar a los requisitos, especificaciones de diseño y código según se van desarrollando.

Debe ser cuidadoso y concienzudo para que sea efectivo. Es más efectivo si se hace intercambiando el objeto a examinar con otro compañero:

Revisión por pares o iguales (peer review):

Consiste en la revisión del código de un programador por otros programadores (sus pares).

Se puede poner en práctica creando un panel que se encarga de revisar periódicamente muestras de código.

2.1.2. Controles estáticos manuales disciplinados

Las revisiones. y 'auditorías son la evolución natural de la comprobación de escritorio; pero a diferencia de aquélla pasan á ser técnicas de grupo. Su misión principal es conseguir que la responsabilidad del control de calidad no recaiga sólo sobre el propio desarrollador.

2.1.2.1. Auditorías

Una auditoría consiste en realizar una investigación para determinar:

- El grado de cumplimiento y la adecuación de los procedimientos, instrucciones, especificaciones, códigos, estándares u otros requisitos de tipo contractual establecidos y aplicables.
- La efectividad y adecuación de la implementación realizada.

Se pueden considerar tres tipos de auditorías:

Auditoría del producto: el objetivo es cuantificar el grado de conformidad del producto con las características requeridas. Las auditorías del producto software más comunes son la auditoría funcional y la auditoría física.

Auditoría del proceso: el objetivo es evaluar el proceso de desarrollo o de gestión, y evaluar su completitud y efectividad, determinando dónde se puede mejorar. En el desarrollo de software se suelen realizar dos tipos de auditorías del proceso:

Auditorías de proyecto: cuyo objetivo es, evaluar la productividad y eficacia del equipo qué trabaja en un proyecto así como la efectividad de bs métodos y herramientas utilizados.

Auditorías de gestión de proyecto: cuyo objetivo es evaluar la efectividad de las prácticas de gestión realizadas y la organización del proyecto.

Auditor(a del sistema de calidad: el objetivo es evaluar la completitud y efectividad del propio sistema de calidad establecido.

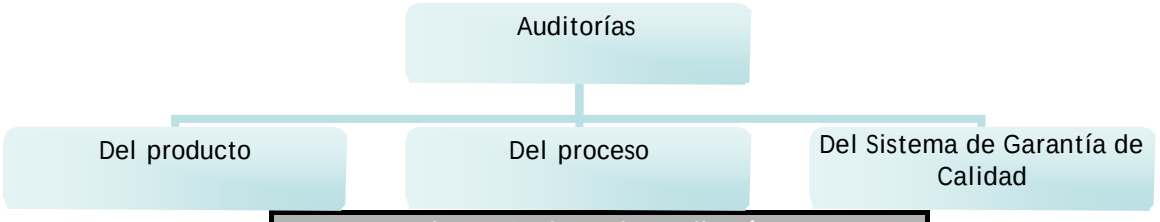


Figura 6 Tipos de auditorías

El procedimiento habitual para realizar una auditoría consta de los siguientes pasos:

1. Planificación: consiste en definir los objetivos de la auditoría y su alcance. En esta etapa se elabora un plan de auditoría, que debería dar respuesta a cuestiones del tipo de las siguientes:
   - ¿Por qué se realiza la auditoría? Puede ser una auditoría de rutina o puede realizarse para resolver problemas concretos.
   - ¿Para qué se realiza? Para mejorar, para conseguir una certificación **...**
   - ¿Cuál es el producto que va a ser auditado?
   - ¿Qué resultados se esperan de la auditoría? En principio, una auditoría debería identificar situaciones problemáticas, tales como desviaciones del estado actual con respecto al estado deseado, y sugerir posibles soluciones o mejoras.
   - ¿Cómo y dónde se van a utilizar los resultados de la auditoría? ¿ Quiénes son los responsables de llevarla a cabo?
   - ¿De qué forma se va a llevar a cabo? Incluyendo una especificación de los datos que se van a recoger y de qué forma se van a recoger.
   - ¿Cuándo se va a realizar?
2. Llevar a cabo la investigación: por lo general, la auditoría se inicia con una reunión de apertura de la investigación, y se lleva a cabo mediante entrevistas y revisiones en las que se recopilan datos.
3. Analizar los datos recogidos: por lo general, el equipo de auditores debe hacer frente a cantidades ingentes de datos, de entre los cuales resulta complicado seleccionar los datos relevantes, por lo que se suelen utilizar técnicas de análisis estadístico. A continuación se realiza una evaluación en paralelo de los resultados por un grupo de evaluadores, se comparan las conclusiones obtenidas y se estudian las causas de las desviaciones significativas.
4. Sugerir soluciones a los problemas encontrados y posibles mejoras.
5. Elaborar y presentar un informe de resultados.

2.1.2.2. Revisiones

Se puede definir una revisión como una reunión formal en la que se presenta el estado actual de los resultados de un proyecto a un usuario, cliente u otro tipo de persona interesada, y se realiza un análisis estructurado de los mismos.

Uno de los objetivos fundamentales de las revisiones técnicas es ofrecer a los gestores información fiable acerca de los aspectos técnicos del proceso de desarrollo de software, de la misma forma que les llega información fiable acerca de los costes y la programación del trabajo, para que con esta información puedan tomar decisiones adecuadas para dirigir con éxito el proyecto.

Con las revisiones se consigue que el peso de la evaluación técnica no recaiga sobre las mismas personas involucradas en la producción del software, que por la posición que ocupan no pueden ser totalmente objetivas, sino en otras personas técnicamente competentes y objetivas.

Las revisiones son, hoy en día, el único método de control de calidad eficaz en las fases iniciales del desarrollo a la hora de identificar desviaciones con respecto a las especificaciones de calidad.

Las revisiones redundan en una mejora directa de la calidad del objeto que se examina y provocan, indirectamente, una mejora de la calidad del proceso de desarrollo, al facilitar

la comunicación entre los miembros del equipo de desarrollo. Al mismo tiempo facilitan el control del coste y el tiempo.

Las diferencias más importantes entre las revisiones y las auditorías son las siguientes:

- Las revisiones se llevan a cabo desde las primeras fases del desarrollo, mientras que las auditorías se llevan a cabo en las fases finales.
- El objetivo de las revisiones es detectar defectos, 'mientras que el objetivo de las auditorías es certificar conformidad e identificar desviaciones.

A) Tipos de revisiones

Hay dos tipos fundamentales de revisiones: las inspecciones y las *walkthrough.* La diferencia entre ellos está en la forma en que se desarrolla la reunión de revisión.

Inspecciones: en las que los participantes van leyendo el documento, paso a paso, guiados por el autor del mismo, y comprobando en cada paso el cumplimiento de los criterios de una lista de comprobación.

Walkthrough (visita guiada): en las que se demuestra la funcionalidad del objeto revisado mediante la simulación de su funcionamiento con casos de prueba y ejemplos. Se introducen al objeto los casos de prueba y se van registrando los resultados intermedios.

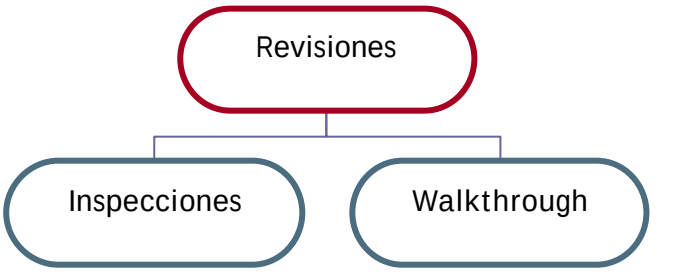


Figura 7 **Tipos de revisiones**

Aunque éstos son los tipos ideales, en la vida real hay otras muchas variantes intermedias, desde las revisiones sin disciplina alguna hasta cualquier tipo de mezcla entre inspecciones y *walkthrough.*

Otras diferencias esenciales entre inspecciones y *walkthrough* son las siguientes:

Las *walkthrough* están planteadas como una medida de ayuda al desarrollador, mientras que las inspecciones están planteadas como una medida de ayuda al gestor.

En las *walkthrough* el objetivo fundamental es incrementar el entendimiento, comprender mejor el objeto, mientras que en las inspecciones el objetivo es detectar defectos.

En las inspecciones el proceso está guiado por la lista de comprobación, y en las *walkthrough* está guiado por la estructura del producto revisado.

Las inspecciones se planifican y procesan de una manera mucho más formal que las *walkthrough.* Se usan para asegurar la satisfacción de los criterios de salida establecidos entre diferentes etapas del desarrollo (revisiones de fase).

La siguiente tabla resume algunas diferencias adicionales entre ellas:

Propiedades	Inspección	Walkthrough
Requiere entrenamiento formal del moderador	Sí	No
Hay unos roles definidos para los participantes	Sí	No
Quién guía la revisión	El moderador	El propietario del producto revisado
Se usan listas de comprobación	Sí	No
Se usan distribuciones por tipo de los errores a buscar	Sí	No
Hay un seguimiento para controlar que la corrección es correcta	Sí	No
Se puede mejorar la eficiencia de la revisión analizando los resultados	Sí	No

B) Fases en una inspección

Una inspección consta de los siguientes pasos:

1. Planificación: la preparación comienza con la selección de los participantes. Debe designarse un coordinador o moderador; un secretario; un presentador, de entre los productores del objeto que se revisa; y otros revisores, que pueden ser desarrolladores, representantes de los usuarios, revisores externos; y posiblemente otros.

   El coordinador es responsable de la planificación de la inspección, de moderar-la reunión (mantener el orden, mantener el foco de la revisión, asegurar que se cubren todos los aspectos necesarios), de preparar el informe final y de realizar el seguimiento y evaluación de las acciones pendientes.

   El secretario es responsable de anotar los elementos de interés (defectos y anomalías descubiertas, acciones pendientes) y ayudar al coordinador en la preparación del informe final.

   - En la planificación es también necesario determinar:
   - Los objetivos de la inspección.
   - Los criterios de finalización de la inspección.
   - El lugar y la fecha para la inspección.
   - La disponibilidad de todos los participantes.
   - La agenda de la reunión.
2. Orientación inicial: es recomendable- realizar una reunión de orientación, previa a la inspección propiamente dicha, cuando se trata de examinar por primera vez un objeto, para dar a los participantes en la revisión una idea del objeto que van a revisar.
3. Preparación individual: con suficiente tiempo antes de la realización de la inspección, se debe hacer llegar a cada revisor una copia de la documentación asociada al objeto que se va a revisar, junto con una lista de comprobaciones o checklist enumerando los posibles defectos que se deben intentar localizar. Una lista de comprobaciones contiene una serie de preguntas, y se supone que al intentar dar respuesta

a estas preguntas saldrán a la luz los problemas que puedan existir. Cada revisor debe completar la lista y anotar cualquier tipo de pregunta o defecto detectado.

4. Reunión de inspección: durante la reunión, el presentador o autor del objeto revisado va guiando al resto a través del mismo para comprobar cada uno de los puntos de la lista de comprobación. Hay varias formas de guiar la reunión:

    - Punto por punto de la lista de comprobación, revisando todo el producto. para cada uno de ellos.
    - Componente a componente del producto, revisando todos los puntos de la lista de comprobación para cada componente.
    - Por grupos de puntos dentro de la lista de comprobación, revisando todo el producto para cada grupo.
    - Cualquier otra solución intermedia.

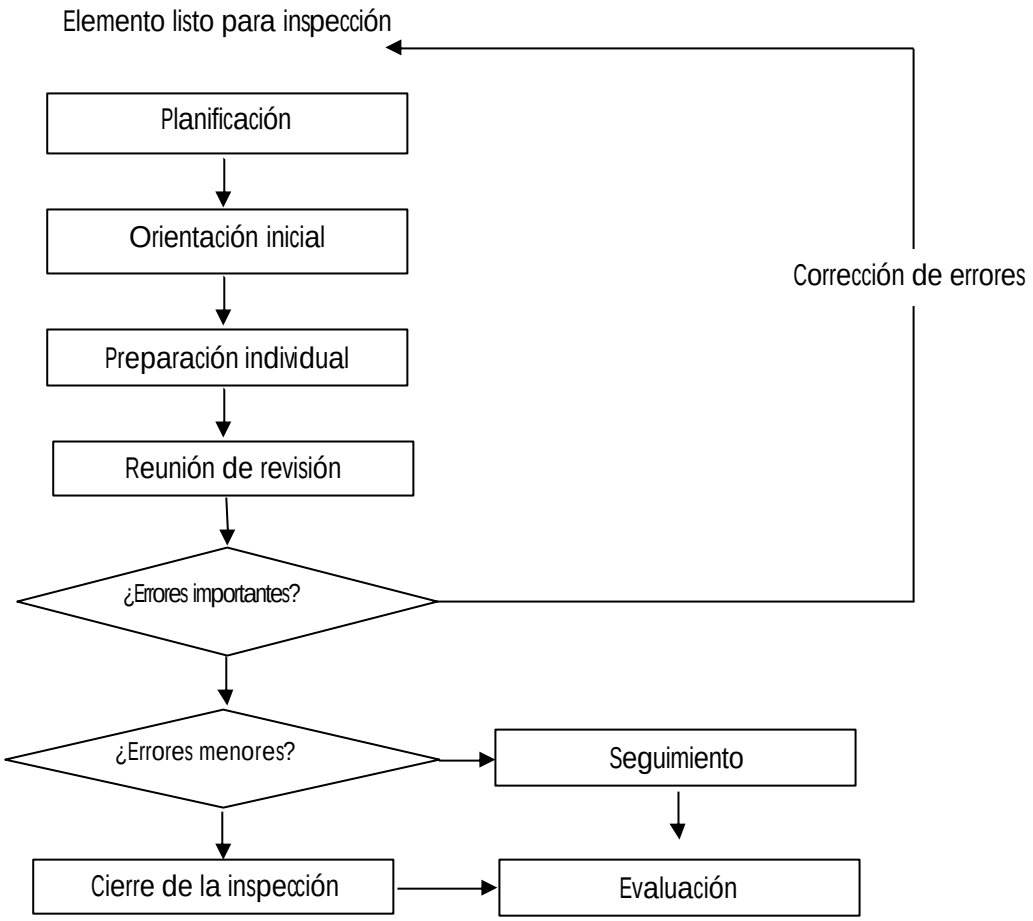
    Los defectos que se detecten durante este proceso se añaden a una lista de acciones pendientes.

    Hay que tener en cuenta que el objetivo de una inspección es descubrir defectos, no corregirlos. Por otro lado, los revisores deben restringirse a los hechos y ser constructivos. Es misión del moderador asegurarse de que esto se cumple.

    Al final de la reunión de inspección, los participantes valoran los resultados de la inspección y se completa un informe. Al finalizar la reunión se pueden producir las siguientes situaciones:

    - Se cierra la inspección sin que se hayan encontrado defectos.
    - Se cerrará la inspección después de que los defectos encontrados se hayan eliminado en la fase de seguimiento.
    - No se cierra la inspección porque se encontraron defectos importantes, y será necesario realizar una nueva inspección.

5. Seguimiento: durante esta fase, el autor del objeto revisado se encarga de conegir los defectos encontrados y generar un informe en el que se especifican las acciones correctivas realizadas para eliminar los distintos defectos.
6. Evaluación: es la última fase y en ella se trata de determinar si se han corregido todos los defectos y si han surgido nuevos problemas durante el proceso de corrección. El moderador se encarga de realizar la evaluación y enviar un informe a la dirección una vez finalizada.



C) Documentos generados en una inspección

Algunos de los informes o documentos que pueden generarse como resultado de una inspección son los siguientes:

- Informe resumen de la inspección: conclusiones breves de la inspección (una página o dos) para la dirección:
    - Qué se ha revisado,
    - Quién lo ha revisado,
    - Cuál fue la conclusión.
- Lista de acciones pendientes: es un informe para los autores del producto revisado explicando qué es lo que está mal y, si puede ser, cómo corregirlo. Necesita ser claro, pero no muy elaborado. Es un documento técnico y transitorio. No debe llegar a la dirección, para no aburrirlos con tantos datos y para que no puedan usar esta información en perjuicio del proceso de inspección.
- Informe de asuntos relacionados: para registrar problemas que salen a la luz durante la inspección pero no están relacionados directamente con el objeto revisado, para que sean notificados a la persona responsable.
- Informe del proceso de inspección: cuando algo ha salido mal en el proceso de inspección en sí mismo.
- Informe final: para informar a la dirección del cierre de la inspección.

D) Fases en una walkthrough

El proceso para realizar una *walkthrough* es mucho más sencillo y consta de tan sólo tres fases:

1. Planificación: similar a la planificación de una inspección, con la diferencia de que no es necesario asignar roles específicos a los participantes; a excepción del presentador, que es quien organiza la *walkthrough* y guía la reunión.
2. Preparación individual: cada revisor examina el objeto revisado, aunque en este caso no se les entregará una lista de comprobación.
3. Reunión de *walkthrough.*

La siguiente tabla resume las fases de inspecciones y w*alkthrough* e indica los objetivos que se persiguen en cada una de ellas:

Etapas	INSPECCIÓN(objetivo)	Walkthrough
1. Orientación	Educación (grupo)	-
2.Preparación individual	Educación (individual)	Educación (individual)
3. Reunión	Encontrar errores (grupo)	Educación (grupo) Discusión de alternativas de diseño Encontrar errores
4. Seguimiento	Arreglar problemas	-
5. Evaluación	Asegurar que todos los problemas se han resuelto correctamente	-

E) Formalidad en las revisiones

Se puede diferenciar también entre las revisiones formales e informales. Se dice que una revisión es formal cuando:

- Es un evento público.
- Se informa por escrito de los resultados.
- Todos los participantes son responsables de la calidad de la revisión.

La ventaja de realizar revisiones formales es que los informes que se generan sirven como hitos para el proyecto, y el hecho de ser algo público promueve una mejor preparación por parte de los participantes. Por contra, la formalidad hace que este tipo de reuniones sean un tanto impersonales.

Para evitar que los participantes se sientan intimidados por el ambiente impersonal y expresen libremente sus opiniones se pueden utilizar técnicas como el *round-robin,* que consiste en ir pasando el turno de uno a otro por todos los participantes en la revisión.

F) Revisiones técnicas y de gestión

Por otro lado, según el objeto que se revise, se suele diferenciar entre las revisiones con orientación técnica y las revisiones orientadas a la gestión (también conocidas como revisiones de proyecto).

Las revisiones técnicas más comunes son:

- Revisión de la especificación de requisitos.
- Revisión del diseño.
- Revisión del código.
- Revisión de las pruebas.
- Revisión del manual de usuario.

Los principales objetivos que se buscan con las revisiones de proyecto, por otro lado, son los siguientes:

- Control de la progresión del proyecto.
- Evaluación de los riesgos asociados al proyecto, con relación al coste, escala de tiempos, recursos utilizados y calidad del producto.
- Evaluación general del producto.

Para que esto sea posible es necesario:

- Que exista un plan de desarrollo bien estructurado, con hitos bien definidos, que permita evaluar la progresión del proyecto.
- Que los resultados del proyectase encuentren bien documentados y hayan sido ya examinados en una revisión técnica.

A continuación vamos a examinar con más detalle las revisiones técnicas más comunes.

G) Revisión de la especificación de requisitos

Este tipo de revisión es muy útil para facilitar el descubrimiento de los errores introducidos en la especificación de requisitos en fases tempranas del desarrollo.

El tipo de errores que se pueden encontrar en este objeto son:

- Requisitos poco claros, contradictorios, erróneos o imposibles de probar.
- Requisitos incompletos o especificación incompleta (faltan requisitos].
- Requisitos irrelevantes para el problema que se trata de resolver.

Algunas de las preguntas que podrían encontrarse en una lista de comprobaciones para la especificación de requisitos son las siguientes:

- ¿Se han especificado todos los recursos hardware necesarios?
- ¿Se han especificado las interfaces externas necesarias?
- ¿Existen contradicciones en la especificación de los requisitos?
- ¿Se han definido los criterios de aceptación para cada una de las funciones especificadas?
- ¿Resulta comprensible la especificación realizada?

H) Revisión del diseño

Se suele diferenciar entre la revisión del diseño preliminar o de alto nivel y la revisión del diseño detallado. El objetivo de estas revisiones es determinar y evaluar el estado en el que se encuentra el proceso de diseño, así como descubrir errores o contradicciones (entre la especificación de requisitos y el diseño o en las interfaces entre módulos).

Algunas de las preguntas que podrían encontrarse en una lista de comprobaciones para el diseño son las siguientes:

- ¿Hay uniformidad en el diseño?
- ¿Se han definido correctamente las interfaces entre módulos?
- ¿Se han definido correctamente las interfaces externas?
- ¿Cubre el diseño todas las funciones incluidas en la especificación de requisitos?
- ¿Cumple el diseño todos los requisitos no funcionales?
- ¿Resulta ambigua la documentación, del diseño?
- ¿Se ha aplicado la notación de diseño correctamente?
- ¿Es el diseño lo suficientemente detallado como para que sea posible implementarlo en el lenguaje de programación elegido?

I) Revisión del código

Las revisiones del código suelen tomar la forma de revisión por pares o inspección, y uno de sus objetivos es determinar que el código se corresponde con el diseño detallado realizado.

Algunos de los aspectos que se examinan en una revisión de código son los siguientes:

- Interfaces.
- Estructura del programa.
- Utilización de variables.
- Fórmulas.
- Entradas y salidas.
- Comentarios
- Cumplimiento a los estándares de codificación.

Las revisiones de código se basan en la lectura del mismo, lo que exige de los programadores un esfuerzo para hacerla legible.
Los estudios realizados demuestran que, mediante las inspecciones de c6dígo, más de la mitad de los errores de programación se pueden detectar antes de pasar a la prueba modular y que los programas que han pasado por una inspección de código contienen considerablemente menos errores.

J) Revisiones de las pruebas

Se pueden efectuar dos tipos de revisiones de las pruebas:

- Revisión del diseño de la prueba.
- Inspección de la prueba.

El objetivo de la revisión del diseño de la pruebas es comprobar que el diseño realizado para la prueba está de acuerdo con los objetivos que se persiguen. Se deben comprobar aspectos como:

- ¿Se han tenido en cuenta todos los objetivos a la horade diseñar los casos de prueba?
- ¿Se han elegido los casos de prueba más adecuados para comprobar la consecución de dichos objetivos?

Los objetivos de las inspecciones de las pruebas, por su parte, son:

- Comprobar que la prueba se ha ejecutado correctamente, de acuerdo con el procedimiento de prueba especificado.
- Análisis de los resultados obtenidos con cada prueba.

2.1.3. Controles estáticos automáticos

Dentro de esta categoría tenemos el análisis estático automático y la verificación formal de programas.

La mayor parte del análisis estático automático del código lo realizan los compiladores, que pueden detectar desde expresiones sintácticamente incorrectas hasta incompatibilidades de tipo y otros errores de tipo semántica.

- Análisis de flujo,
- Ejecución simbólica.

2.1.3.1. Análisis de flujo

Se basa en una representación gráfica. Se usan grafos en los que los nodos representan sentencias o segmentos de programa y los arcos posibles transiciones de control desde un segmento a otro.

Estos grafos se pueden utilizar para identificar caminos, para analizar el comportamiento del programa, para situar puntos de ruptura y para otras técnicas de análisis estático.

La técnica de análisis de flujo de datos fue introducida por Cockey Allen, y consiste en trazar el comportamiento de las variables del programa desde su inicialización hasta que termina la ejecución del programa.

Ya clasificando las variables en:

Referenciadas: Cuando se obtiene su valor en memoria durante la evaluación de una expresión en una sentencia (parte derecha de una sentencia de asignación, Índice de acceso a un array, etc.).

Definidas: cuando se obtiene un nuevo valor para la variable tomo consecuencia de la ejecución de una sentencia (parte izquierda de una sentencia de asignación).

No referenciadas: cuando ya no se puede determinar el valor de la variable a partir del flujo del programa (variables locales fuera de la subrutina en la que se definen, etc.).

Se van asociando valores, en cada nodo del grafo de flujo de datos, a unos *tokens* que representan a las distintas variables del programa. Estos valores indican si la variable' queda referenciada (r), definida (d) o no referenciada (n) con la ejecución de la sentencia representada en ese nodo, o simplemente no la afecta (1).

A partir de estos valores se generan «expresiones de camino» para cada variable, concatenando los valores asignados a los *tokens* de dicha variable en nodos consecutivos.

Se analizan entonces las expresiones de camino para detectar anomalías:

...dd... se ha definido dos veces sin ser referenciada,

...nr... se ha referenciado sin haberla definido previamente,

r... se ha referenciado sin haberla definido previamente, etc.

2.1.3.2. Ejecución simbólica

Consiste en la ejecución simbólica de ciertos caminos dentro del programa, durante la cual se verifican, ciertas expresiones simbólicas con respecto a ciertas condiciones y afirmaciones preestablecidas.

Dichas expresiones simbólicas están compuestas por nombres de variables, en lugar de valores concretos para dichas variables. Todas las manipulaciones sobre dichas variables durante la ejecución, y todas las condiciones, se manejan deforma simbólica. Todos los pun-

tos de decisión quedan indeterminados puesto que no se conocen los valores de las variables.

El resultado de la ejecución simbólica se puede entonces comparar con la especificación externa del programa y ver si están de acuerdo una con otra.

El resultado de una ejecución simbólica es normalmente una expresión larga y compleja. La mejor forma de analizarla es descomponerla en una estructura de árbol, donde cada hoja representa un camino de ejecución y cada ramificación representa un punto de decisión en el programa.

El árbol resultante se puede usar también como ayuda para generar casos de prueba.

Aunque esta técnica puede ser muy útil, también tiene algunos problemas:

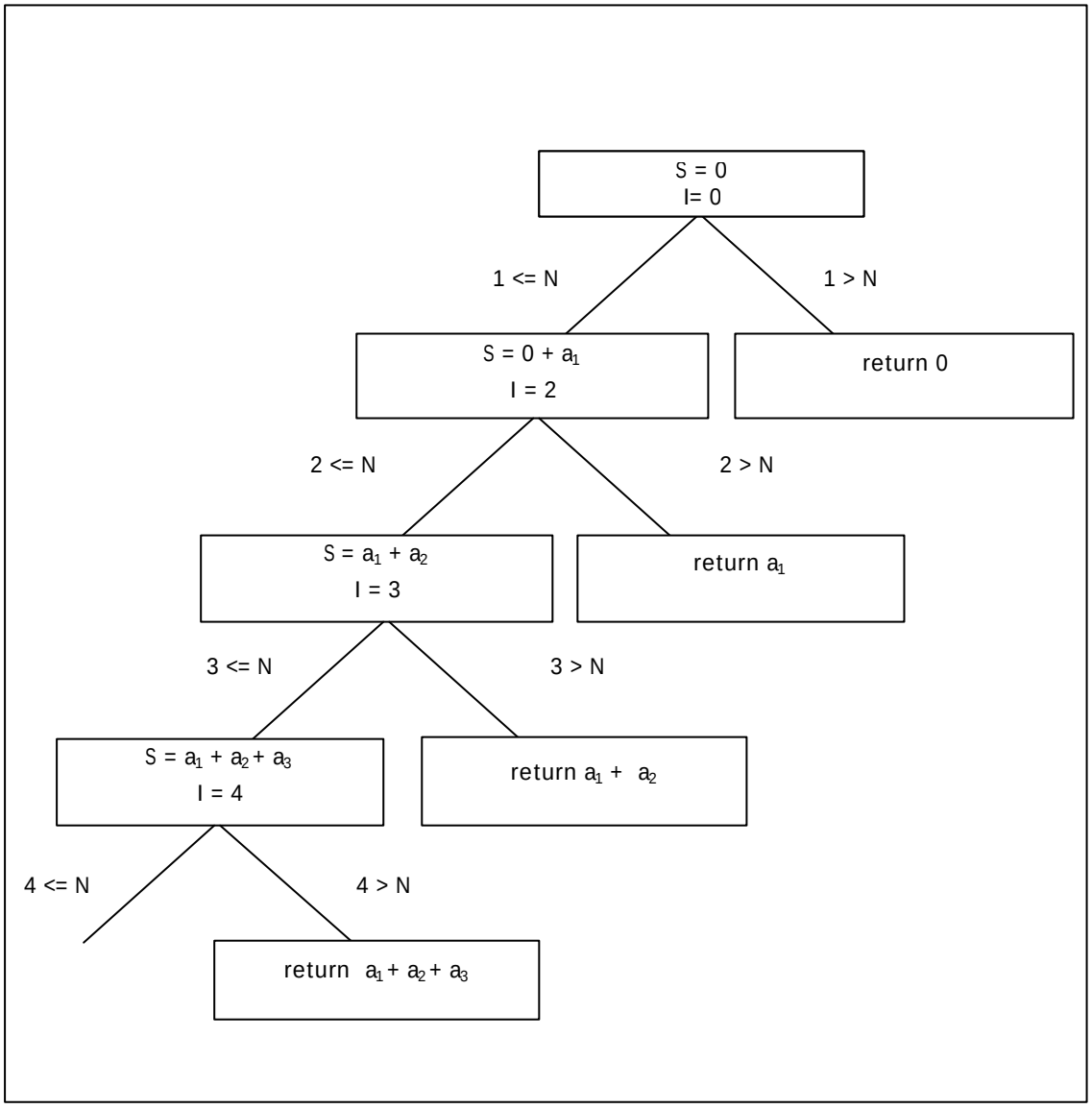
- Si el programa no tiene ciclos, el árbol resultante es finito, pero si tiene ciclos el árbol es infinito. Lo que se hace es tomar un número finito de iteraciones.
- Al utilizarlo para generar casos de prueba nos enfrentamos aun caso de indecidibilidad.

Ejemplo

Función en Ada que calcula la suma de los N elementos de un array entero A:

```
Function SUM (A: INTARRAY; N: NATURAL) return INTEGER is
        S: INTEGER := O;,
        1: NATURAL:= 1;
Begin
        while I <= N loop
                S: = S +A(I);
                1: = I + 1;
        end loop;
        return (S);
End SUM;
```

Resultado de la ejecución simbólica:



2.1.3.3. Verificación formal

Consiste en demostrar matemáticamente la corrección de un programa con respecto a sus especificaciones.

Para ello, se considera el programa como un objeto formal, es decir, como una cadena de un lenguaje formal, con una sintaxis y una semántica formal. Es también necesario que la especificación se haya escrito en algún lenguaje formal. Por eso no siempre es posible realizar este tipo de verificación.

Por lo general, esta técnica sólo se utiliza para sistemas críticos, debido al coste que conlleva.

Los métodos de verificación formal de programas más conocidos son los basados en la lógica de Hoare [Hoare, 1969], los basados en la aproximación de Dijkstra [Dijkstra, 1989] y la aproximación funcional de Mills [Mills, 1987].

2.2. Controles Dinámicos

Definición: Se llama controles dinámicos a aquellos que requieren la ejecución del objeto que se está probando o de un modelo del mismo.

Hasta la fecha no se ha desarrollado ninguna teoría universalmente aceptada acerca de la prueba de software. Lo único que hay es un conjunto de aproximaciones metodológicas que facilitan y hacen más eficiente el proceso de prueba.

Definiciones: Se llama prueba del software al proceso en el que se ejecuta un sistema con el objetivo de detectar fallos.

Se llama depuración al proceso en el que se localiza el defecto que es la causa de un fallo, se determina la forma de corregirlo, se evalúa el efecto de la corrección y se lleva a cabo la corrección. Por lo general, después del proceso de depuración será necesario repetir el proceso de prueba, para garantizar que el defecto quedó efectivamente corregido y que no se introdujeron nuevos defectos.

El coste de detección de los defectos suele ser mucho mayor que el coste de corrección de los mismos, y éste es un punto en contra de las pruebas como técnica de control de calidad, ya que siempre es necesario un paso de diagnostico hasta que se localiza la causa de los fallos. En otras actividades de control de calidad, por el contrario, como pueden ser las revisiones, se localizan directamente los defectos, no sus síntomas, por lo que nos ahorramos el proceso de diagnóstico.

En un proyecto grande la prueba se puede llevar hasta el 50 o 60 por 100 del esfuerzo dedicado al proyecto. Por eso es muy importante seleccionar bien las pruebas que se van a realizar, teniendo en cuenta que sólo las pruebas que revelan defectos son las que realmente merecen la pena. El objetivo del proceso de prueba no es, como pudiera parecer, demostrar que el software está libre de defectos, sino precisamente descubrir defectos. Por ello, se deben seleccionar especialmente aquellos casos de prueba que incidan en las secciones del programa más complejas, en los valores límite de las variables yen la tolerancia a fallos del diseño.

Aunque la prueba es una parte importante del control de calidad, es importante darse cuenta de que no es la única.

A continuación veremos cuáles son las actividades que es necesario realizar para probar un sistema software, y cuáles son los principales métodos de prueba que se pueden utilizar.

2.2.1. Tipos de pruebas

El proceso de prueba conlleva la realización de un conjunto de tareas a lo largo del ciclo de vida del sistema.

De acuerdo con el estándar iEEE 1012-1986, el conjunto mínimo de pruebas que se deben realizar son:

- Prueba modular, prueba unitaria o prueba de componentes.
- Prueba de integración.
- Prueba del sistema.

- Prueba de aceptación.

También se suele realizar otro tipo de prueba llamada prueba de regresión.

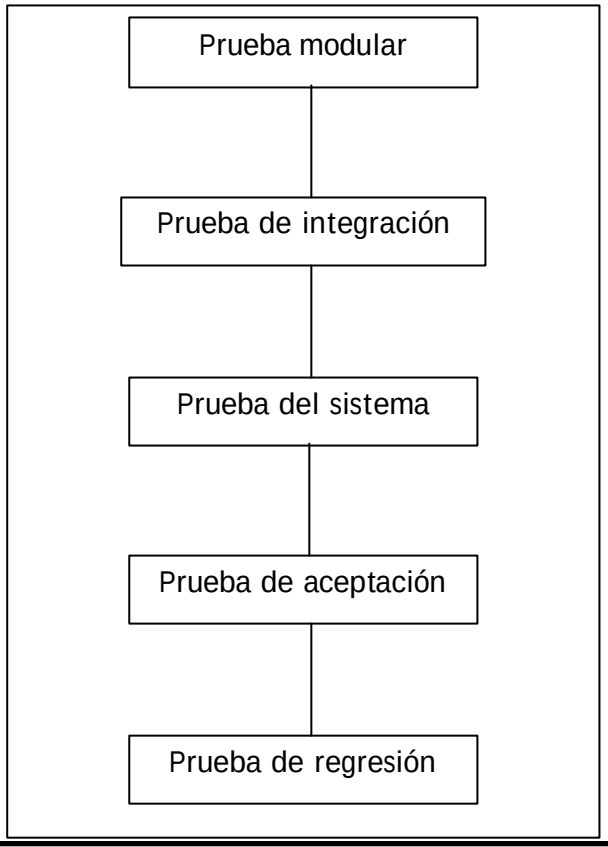


Figura 9 **Tipos de pruebas**

2.2.1.1. Prueba modular

La prueba modular consiste en la prueba de cada módulo aislado del resto del sistema.

2.2.1.2. Prueba de integración

Se realiza a medida que los diferentes módulos del sistema se integran en el mismo.

Ya se ha realizado la prueba modular, y se supone que todos módulos son correctos: El objetivo fundamental de esta prueba es comprobar que las interfaces entre los distintos módulos son correctas. Algunas de las comprobaciones que es necesario realizar son:

Corrección en la sintaxis en la invocación de procedimientos y funciones.

Compatibilidad de tipos entre los argumentos del procedimiento o función y los parámetros de llamada.

Corrección y completitud de las especificaciones de los módulos.

Se pueden utilizar tres posibles estrategias de integración:

De arriba a abajo *(top-down):* consiste en empezar la integración y la prueba por los módulos que están en los niveles superiores de abstracción, e integrar incrementalmente los niveles inferiores.

De abajo a arriba *(bottom-up):* consiste en empezar la integración y la prueba por los módulos que están en los niveles inferiores de abstracción, e integrar incrementalmente los niveles superiores.

De *big-bang*: consiste en integrar y probar todo al mismo tiempo.

2.2.1.3. Prueba del sistema

La prueba del sistema se realiza cuando se han integrado todos los módulos, y su objetivo es comprobar que el sistema satisface los requisitos del usuario, tanto los funcionales como los no funcionales.

2.2.1.4. Prueba de aceptación

La prueba de aceptación se realiza una vez que el sistema se ha implantado en su entorno real de funcionamiento, y su objetivo es demostrar al usuario que el sistema satisface sus necesidades.

2.2.1.5. Prueba de regresión

La prueba de regresión tiene como objetivo comprobar que toda nueva versión de un producto software es de no menos calidad que la versión anterior, es decir, que al introducir cambios no se ha reducido la valoración de ninguna de las características de calidad que tenía el producto.

2.2.2. Métodos de prueba

A continuación vamos a vedas dos grupos en que se clasifican los métodos de prueba:

- Métodos de caja negra.
- Métodos de caja blanca.

2.2.2.1. Métodos de caja negra

En este tipo de métodos, el objeto que se desea probar se ve como una caja negra. Esto quiere decir que la elección de los casos de prueba no se va a basar en el conocimiento que se tenga acerca de la estructura del objeto, sino en el conocimiento acerca de la funcionalidad deseada (descripción funcional).

A la prueba de caja negra también se le llama prueba funcional o prueba orientada al diseño.

Una prueba de caja negra exhaustiva requeriría la generación de un caso de prueba, por cada combinación posible (válida o no válida) de los valores de entrada, lo cual resulta imposible en la mayor parte de los casos por producirse una explosión combinatoria. Por eso se utilizan diferentes criterios a la hora restringir el conjunto de casos de prueba.

Los métodos de selección del conjunto de casos de prueba más usuales son:

- Método de clases de equivalencia:

  Consiste en dividir las posibles entradas al sistema en clases de equivalencia, de tal forma que todos los miembros de una misma clase de equivalencia prueben las mismas propiedades en el sistema, por lo que sólo va a ser necesario seleccionar un elemento de cada clase de equivalencia.

- Análisis de valores frontera o valores límite:

  Consiste en seleccionar como casos de prueba aquellos valores de entrada que caen en la frontera de las clases de equivalencia (justo a un lado, justo al otro y justo en la frontera).

- Grafos causa/efecto y tablas de decisión:
  Consiste en crear un grafo causa/efecto a partir de las especificaciones, y selec-

cionar suficientes casos de prueba como para asegurar la cobertura del grafo. Se llama causas a las características de los datos de entrada y efectos a las clases de salidas que puede proporcionar el programa. A partir del grafo causa/efecto se construye una tabla de decisión que refleje las dependencias entre causas y efectos. Lo que se hace entonces es reducir la tabla de decisión y seleccionar sólo un caso de prueba para todas las causas que producen el mismo efecto, o para cada columna de la tabla de decisión.

- Adivinación de errores:
  Consiste en tratar de imaginar cuáles son los errores que se pueden haber cometido con mayor probabilidad, y generar casos de prueba para comprobar dichos errores.

2.2.2.2. Métodos de caja blanca o caja transparente

En este tipo de métodos, el objeto que se desea probar se ve como una caja blanca. Esto quiere decir que la elección de los casos de prueba se va a basar en el conocimiento que se tenga acerca de la estructura del objeto (diseño detallado, diagramas de flujo de datos y de control, código fuente).

A la prueba de caja blanca también se le llama prueba estructural.

Los métodos de caja blanca se pueden clasificar, a su vez, en dos grupos:

- Los basados en métricas de cobertura.
- Los basados en métricas de complejidad.

A) Métodos basados en métricas de cobertura

Todo programa se puede representar mediante un grafo de flujo de control, donde cada nodo es una sentencia o una secuencia de sentencias. Los arcos dirigidos en el grafo representan el flujo de control.

Para cada conjunto de datos de entrada el programa se ejecutará a través de un camino concreto dentro de este grafo. Cuando el programa incluye estructuras iterativas, el número de posibles caminos en el grafo puede ser infinito.

Una prueba de caja blanca exhaustiva requeriría la generación de un caso de prueba por cada posible camino. Como esto no es posible, por lo general, se utilizan métricas que dan una indicación de la calidad de un determinado conjunto de casos de prueba en función del grado de cobertura del grafo que consiguen. Las métricas más utilizadas son:

- Cobertura de sentencias.
- Cobertura de segmentos entre decisiones.
- Cobertura de decisiones de ramificación.
- Cobertura de condiciones.
- Cobertura de todas las combinaciones de condiciones.
- Cobertura de caminos

B) Métodos basados en métricas de complejidad

Las métricas de complejidad más utilizadas en la generación de casos de prueba son las de MacCabe:

- Complejidad ciclomática (Arcos - Nodos + 2 x Número de componentes conexos).
- Complejidad esencial (Complejidad ciclomática – Número de subgrafos propios de entrada y salida única).
- Complejidad real (número de caminos ejecutados).

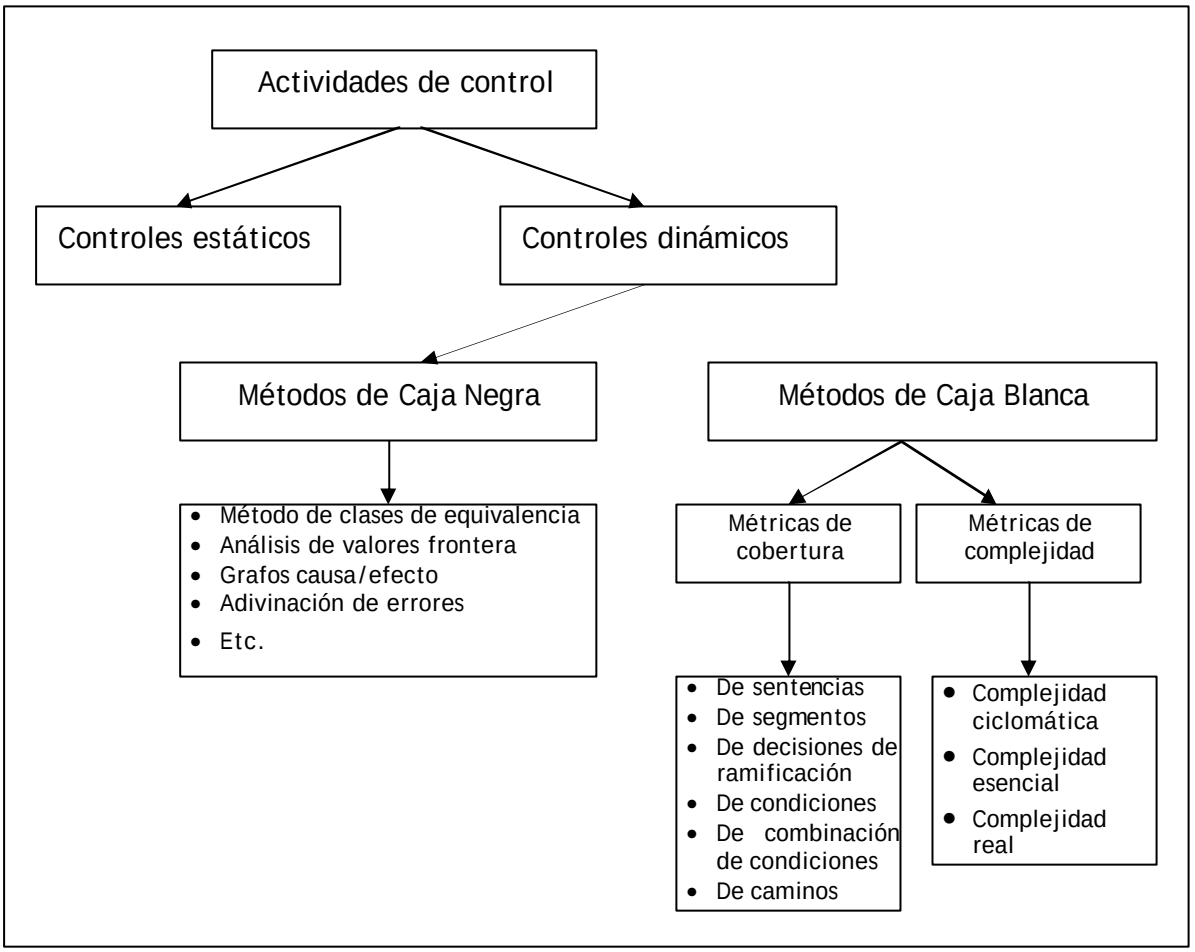


Figura 10 **Tipos de controles dinámicos**

2.2.3. Metodología de prueba

Cada uno de los diferentes tipos de prueba implica la realización de un conjunto de actividades estándar, así como la producción de un conjunto de salidas estándar.

Actividades Estándar de Prueba	***Salidas Estándar Asociadas***
Planificación de la prueba	Plan de pruebas
Diseño de la prueba	Documento de diseño de la prueba
Determinación de los casos de prueba	Especificación de los casos de prueba
Planificación del procedimiento de prueba	Especificación del procedimiento de prueba
Ejecución de la prueba	Informe de los casos de prueba
Análisis y evaluación de la prueba	Informe de la prueba

A continuación vamos a ver cada una de estas actividades en detalle.

1ª. La Planificación de la prueba.

Esta actividad consiste en la creación de un plan de pruebas en el que se registra:

- El objetivo del proceso de prueba.

- Los objetos que hay que probar.
- Las características que se van a probar y las que no.
- El método de prueba a utilizar.
- Los recursos que se van a emplear.
- El plan de tiempos.
- Los productos a generar durante las pruebas.
- El reparto de las responsabilidades.

2ª. Diseño de la prueba

Esta actividad consiste en dar instrucciones detalladas acerca de:

- Cómo llevar a cabo la prueba para alcanzar los objetivos deseados.
- De qué forma se van a utilizar los métodos de prueba.
- Qué objetos se van a probar en cada una de las pruebas.
- Qué criterios se van a utilizar para determinar si el objeto pasa o no la prueba.

3ª. Determinación de los casos de prueba.

Esta actividad consiste en especificar el conjunto de casos de prueba a utilizar en función del diseño realizado para la prueba. Para cada caso de prueba habrá que especificar:

- Qué objetos se van a probar.
- Qué entradas se les van a dar.
- Cuáles son las salidas esperadas.

4ª. Planificación del procedimiento de prueba.

Esta actividad consiste en fijar un conjunto de pasos para la ejecución de la prueba. Se especifica detalladamente:

- La secuencia exacta de ejecución de los distintos casos de prueba.
- Los requisitos que hay que cumplir para la ejecución de cada caso.
- Las condiciones de terminación de cada uno de ellos.

5ª. Ejecución de la prueba.

Esta actividad consiste en ejecutar cada caso de prueba, según el procedimiento especificado en el paso anterior, y registrar los incidentes o problemas encontrados durante la misma.

6ª. Análisis y evaluación de la prueba.

Se examinan los resultados de la prueba y se decide si se han alcanzado los objetivos propuestos o se debe repetir la prueba.

3 Actividades de Garantía de Calidad del Software

3.1. ¿Qué es la garantía de Calidad?

Reifer [Reifer, 1988] define la garantía de calidad como «un conjunto de actividades de planificación, estimación y supervisión de las actividades de desarrollo, que se realizan de forma independiente al equipo de desarrollo, de tal forma que los productos software resultantes cumplen los requisitos establecidos».

Bersoff la define como «un conjunto de procedimientos, técnicas y herramientas, aplicados por profesionales, durante el ciclo de desarrollo de un producto, para asegurar que el producto satisface o excede los estándares o niveles de calidad preestablecidos».

Así pues, como compendio de las definiciones anteriores, se puede decir que la garantía de calidad abarca todas aquellas actividades o prácticas que se realizan con el objetivo de asegurar un cierto nivel de calidad en el producto desarrollado.

También es interesante la puntualización que encontramos en la primera definición: «que se realizan de forma independiente al equipo de desarrollo». Por lo general, el equipo de garantía de calidad es diferente del equipo de desarrollo, especialmente en proyectos grandes. En cuanto al tamaño de este equipo, sirva como indicación que el promedio está en una persona de garantía de calidad por cada 15 a 40 personas en el equipo de desarrollo.

Las áreas que caen bajo la responsabilidad del grupo de garantía de calidad son tres:

1ª. Las metas y objetivos: debe asegurar que las metas de la organización, en primer lugar, y los objetivos del usuario, en segundo lugar, se están satisfaciendo, y que no existen conflictos entre ellos, o entre los objetivos de diferentes usuarios.

2ª. Los métodos: debe asegurar que las actividades de desarrollo de software siguen los procedimientos establecidos, se ajustan a los estándares seleccionados, están de acuerdo con las políticas de la organización y se ejecutan según las guías de trabajo y recomendaciones disponibles.

3ª. Rendimiento: debe asegurar que se optimiza la utilización del hardware y software en los productos desarrollados, que son económicos (se desarrollan con el menor coste posible), eficientes (sacan el máximo partido posible a los recursos utilizados) y efectivos (alcanzan el resultado deseado con la menor cantidad posible de recursos, tiempo y esfuerzo).

El ámbito del control de calidad es el producto software, mientras que el ámbito de la garantía de calidades también el proceso de desarrollo. La relación entre ellos es doble. Por un lado, el grupo de garantía de calidad es responsable de definir el tipo de control de calidad que se va a realizar, pero por lo general no es responsable de efectuar personalmente dichos controles, sino que estos serán realizados por los miembros del equipo de desarrollo. Por otro lado, el grupo de garantía de calidad va a utilizar los resultados del control de calidad para evaluar y mejorar el proceso de desarrollo, para conseguir productos de más calidad.

Las principales tareas del grupo de garantía de calidad, por lo tanto, son:

1ª. Planificación de la calidad: consiste en seleccionar, clasificar y ponderar las propiedades de calidad que se van a establecer como requisitos, con respecto al producto y con respecto al proceso. Se elegirán también los mecanismos de control de calidad a utilizar para medir y evaluar estas características y se determinarán las metas a alcanzar.

2ª. Supervisión de la calidad: consiste en supervisar y corregir, si es necesario, el trabajo que se está realizando (según los resultados obtenidos con las actividades de control de calidad), con el objetivo de llegar a satisfacer los requisitos establecidos.

3ª. Construcción de la calidad: actividades constructivas son aquellas que sirven para «construir» la calidad, es decir, son actividades preventivas cuyo objetivo es evitar la introducción de errores mediante la puesta en práctica de ciertos principios, métodos, formalismos y herramientas.

3.2. Actividades contractivas de Garantía de Calidad

Se pueden considerar diferentes tipos de actividades constructivas de garantía de calidad:

- Técnicas: incluyen, por ejemplo, la aplicación de principios, técnicas y herramientas de Ingeniería de Software.
- Organizativas: incluyen, por ejemplo, la aplicación de modelos de proceso o planes.
- Humanas: incluyen, por ejemplo, la formación del personal y la motivación.

A continuación vamos a ver algunos de los elementos más importantes para la construcción de calidad.

3.2.1. Los modelos de proceso software

La aplicación de modelos de proceso software es una de las actividades constructivas de garantía de calidad más importantes, junto con los métodos y herramientas.

Un modelo de proceso software es una idealización del proceso de desarrollo y mantenimiento de software.

Un modelo de proceso descompone el proceso de desarrollo en una serie de fases y da una descripción de las distintas actividades que se realizan en cada fase y de los resultados que se obtienen.

Las actividades de cada fase se van a llevar a cabo con la ayuda de otras medidas constructivas de garantía de calidad, como son los principios, los métodos, las herramientas ó los ejemplos.

3.2.2. Los métodos y formalismos

La utilización de métodos tiene la ventaja, desde el punto de vista de la garantía de calidad, de que la sistematización de los procedimientos facilita la prueba de los resultados obtenidos.

Además, según algunos estudios realizados, se producen incrementos en la productividad de entre un 50 y un 150 por 100, según el método empleado.

Algunos ejemplos concretos de métodos son:

- El análisis y diseño estructurado y la programación estructurada.
- La aplicación de métodos de estimación de riesgos.
- La aplicación de métodos de estimación de coste y esfuerzo.
- La aplicación de métodos de especificación de requisitos.

A menudo los métodos se asocian con determinados formalismos o lenguajes formales. Así, por ejemplo, para el análisis estructurado existen los formalismos de De Marco [DeMarco, 1979] o de Yourdon [Yourdon. 1989].

Por otro lado, la utilización de formalismos suele ser un prerrequisito para poder automatizar las pruebas.

3.2.3. Las herramientas y entornos de desarrollo

Las herramientas CASE *(Computer Aided Software Engineering)* son herramientas informáticas que facilitan la producción de software.

Se pueden distinguir varios tipos de herramientas CASE, y la utilización de cada una de ellas contribuye a asegurar la construcción de productos de calidad:

- Herramientas de planificación de sistemas de información.
- Herramientas de análisis y diseño. Incluyen, en esencia, editores de texto y de gráficos para la creación de especificaciones, herramientas de prototipado y un repositorio. Suelen incorporar algunas funciones de garantía de calidad, como son:
  - Comprobación de la sintaxis y la semántica de la especificación.
  - Comprobación de consistencia y completitud de la especificación.
  - Comprobación de ciertas características de calidad.
  - Seguimiento de los requisitos a través de los diferentes documentos del ciclo de vida del producto.
- Entornos de programación (herramientas de prototipado, preprocesadores, compiladores, editores, herramientas de depuración, generadores de aplicaciones...).
- Herramientas de pruebas. Son herramientas que automatizan algunos aspectos del proceso de prueba. Las más utilizadas son los generadores de casos de pruebas, las herramientas de ejecución automática de los casos de prueba, las de monitorización de los resultados de las pruebas y las de generación de informes.
- Herramientas de ayuda al mantenimiento o de reingeniería.
- Herramientas de gestión de proyectos.
- Herramientas de gestión de configuraciones.

3.2.4. Los lenguajes de programación

La importancia del lenguaje de programación elegido, en términos de la calidad del producto, se hace patente, sobre todo, en la fase de mantenimiento.

Algunas de las características que facilitan la creación de software de calidad son:

- El concepto de módulo, con una separación clara entre la interfaz del módulo y su contenido.
- La compilación separada, que permite detectar, en tiempo de compilación, posibles errores en las interfaces entre módulos.
- Los tipos abstractos de datos y el ocultamiento de información, que permiten separar la representación de los datos de su utilización.
- Flujo de control estructurado, que conduce a estructuras de programas más claras y fáciles de probar.
- Las comprobaciones de tipos de datos en tiempo de ejecución.
- La utilización de nombres significativos para los programas y los diferentes elementos del programa.
- La programación orientada a objetos, que implica ocultamiento de información, abstracciones de datos, herencia y enlace dinámico, porque facilita la modificación, extensión y reutilización de código.

3.2.5. La documentación

Para poder realizar las tareas propias de la garantía de calidad, la documentación juega un papel esencial. Si no se dispone de suficiente documentación o la documentación no es adecuada, no se va a poder comprobar si se satisfacen o no los requisitos.

Un aspecto, por lo tanto, que también es importante tener en cuenta, es la calidad de la documentación. Algunas de las características de calidad que se pueden considerar para la documentación son:

- Facilidad de modificación.
- Consistencia: hasta qué punto están de acuerdo el estado actual del objeto que se describe en la documentación y la descripción que de él se hace.
- Claridad.
- Identificación adecuada: hasta qué punto el índice es claro y permite al lector localizar la información que le interesa.
- Conformidad con los estándares de documentación.
- Comprensibilidad: hasta qué punto el documento hace llegar al usuario la información que pretende transmitir.
- Completitud.
- Ausencia de contradicciones.

3.2.6. Los factores humanos

No hay que olvidar que en el proceso de desarrollo de software los factores humanos son muy importantes y juegan un papel decisivo en la construcción de sistemas de calidad.

Algunos factores que es necesario considerar son:

- La cultura de la organización.
- La comunicación entre los miembros del equipo.
- El entorno físico de trabajo.
- La formación.
- La motivación.
- La dirección.
- El liderazgo.
- Etcétera.

3.2.7. Otros

Otros ejemplos de prácticas y factores que contribuyen a construir la calidad en el producto que se está desarrollando son:

- Los estándares y convenciones: es importante que se establezcan y que se compruebe su seguimiento.
- Los ejemplos y patrones: son muy útiles, especialmente cuando no se tiene la suficiente experiencia en algún tema.
- La gestión de configuración.
- La gestión de problemas
- El control del código: es importante controlar el almacenamiento y mantenimiento de versiones del código.

3.3. El Coste de La Calidad

El coste necesario para conseguir productos de calidad tiene dos componentes:

- Prevención de errores: construcción de la calidad.
- Detección de defectos: control de calidad.

Por el contrario, si no invertimos suficiente dinero y esfuerzo en la construcción y control de la calidad, obtendremos productos de baja calidad, y esto también tiene su coste. El coste de la no calidad también tiene dos componentes:

El coste de corrección de los defectos que se vayan poniendo a la luz.

- Repercusiones externas: falta de credibilidad, descontento de los usuarios, responsabilidad civil, pérdida de clientes.

Es importante darse cuenta de que a la larga acaba saliendo más caro desarrollar productos de baja calidad que productos de calidad.

Para tratar de cuantificar el coste de corrección de los defectos, veamos algunos datos estadísticos:

- Distribución de defectos detectados según la fase en la que se introdujeron:

  Especificación 56%
  Diseño 27%
  Codificación 7%
  Otros 10%

- Porcentaje, sobre el coste total de corrección de los defectos, según la fase en la que se introdujeron:

  Especificación 82%
  Diseño 13%
  Codificación ..... 1 %
  Otros 4%

- Coste de corrección según la fase en la que se detectaron (en $):

  Análisis 200
  Diseño 500
  Codificación 1200
  Pruebas 5000
  Implantación 15000

Vemos cómo la mayor parte de los defectos se introducen en las fases iniciales del proceso de desarrollo. También vemos que los errores tempranos salen mucho más caros que los errores que se introducen en la codificación o las pruebas, y que cuanto más tardamos en detectarlos más caros nos salen.

Una pequeña inversión en actividades de control de calidad durante las fases iniciales del proceso puede suponer una reducción drástica del coste de los defectos que introducimos. Igualmente, una pequeña inversión en actividades constructivas de calidad supondrá una reducción en el número de defectos introducidos y un ahorro considerable.

4 Gestión de Calidad del Software

4.1. ¿Qué es un Sistema de Calidad?

El tipo y número de actividades de garantía de calidad que es necesario adoptar en un proyecto o en una organización concreta depende mucho del tamaño y complejidad de los productos software que se estén desarrollando.

También influyen otros muchos factores, como pueden ser el tipo de proceso de desarrollo de software que se utiliza, la estructura organizativa de la empresa, la motivación del personal, los métodos y herramientas que se estén utilizando, etc.

El sistema de calidad es el que define cómo implementar la garantía de calidad. Es un marco en el que se establecen las diferentes estrategias, actividades y herramientas de garantía de calidad que se van a utilizar.

Se puede definir un sistema de calidad en tres niveles diferentes:

- Organización: es éste el nivel en el que normalmente se establece el sistema de calidad.
- Proyecto.
- Fase de desarrollo.

El sistema de calidad establece también de qué forma se reparten las tareas v responsabilidades de garantía de calidad entre las diferentes unidades organizativas de la empresa y el personal. La forma en que esto se realice dependerá mucho de la estructura organizativa de la empresa y aún no existe un modelo generalmente aceptado de cómo hacerla.

Es también necesario especificar cómo integrar las diferentes tareas de garantía de calidad en el modelo de proceso de desarrollo de software que sigue la organización o el proyecto en cuestión. Esta integración quedará documentada en un Plan de Garantía de Calidad, que será uno o más de los distintos planes que se elaboren para cada proyecto. En este Plan se deben identificar los diferentes criterios de calidad que se van a considerar en cada fase del desarrollo, tanto los relativos al producto como los relativos al proceso, así como los métodos y recursos que se van a utilizar para comprobar dichos criterios, y los informes que serán necesario producir.

Toda esta información debe quedar recogida en un conjunto de documentos.

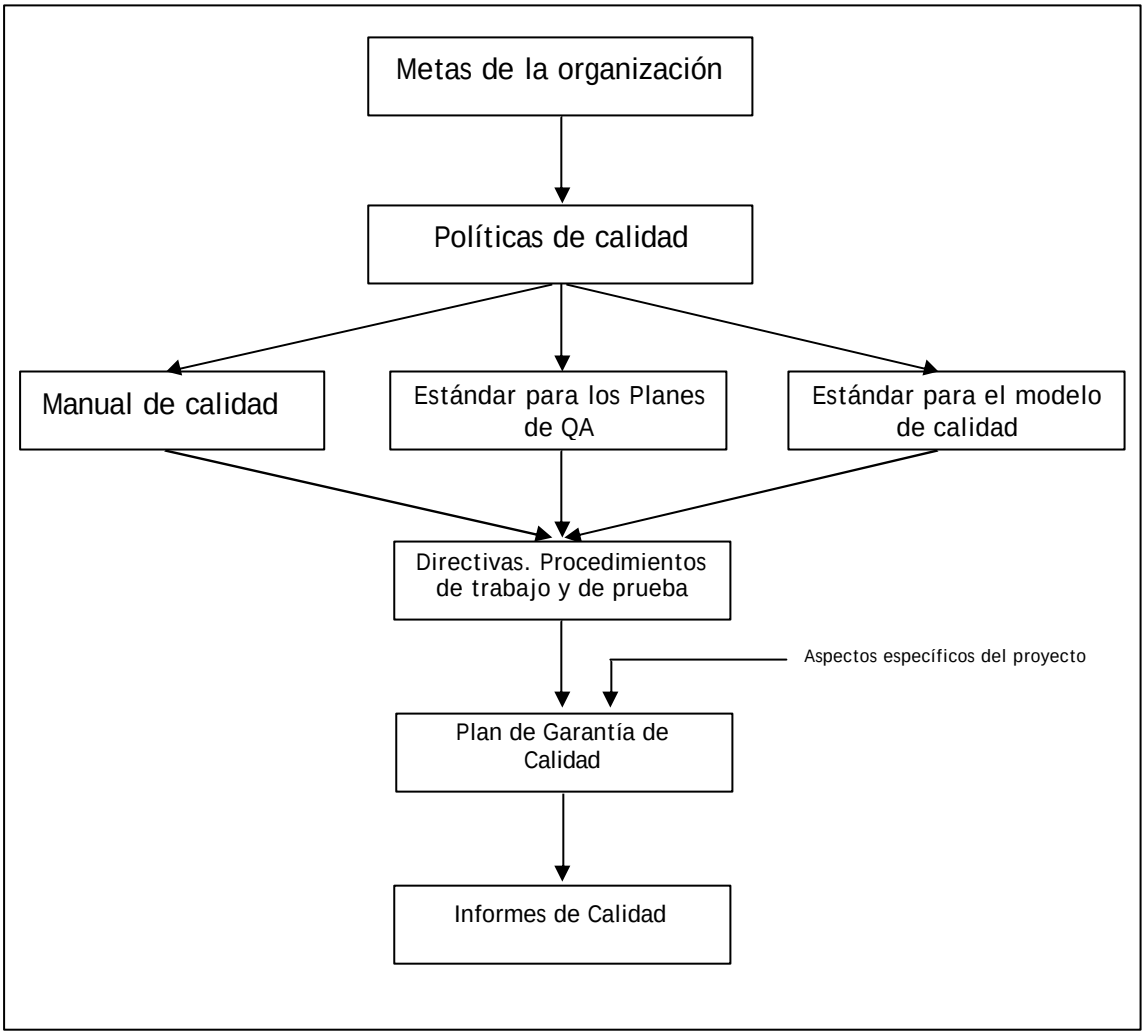


Figura 10 **Tipos de controles dinámicos**

El Manual de Calidad debe ser una guía al sistema de calidad. Debe especificar la terminología, políticas, principios, responsabilidades v procesos del sistema, así como los estándares en los que se basa (por ejemplo, ISO 9001, 9002 o 9003). Debe dar respuesta a las cuestiones de quién, dónde y por qué la garantía de calidad.

Es también necesario adoptar un estándar para la elaboración de Planes de Garantía de Calidad, como por ejemplo el IEEE Std 730-1989.

Es conveniente, asimismo, basar la garantía de calidad en un modelo de calidad reconocido, que ayude en la selección de propiedades y métricas de calidad a utilizar.

Las directivas y los procedimientos de trabajo y de prueba van a facilitar la selección y la puesta en práctica de actividades de garantía de calidad. Vienen a contestar la cuestión de qué y cómo aplicar la garantía de calidad.

Para cada proyecto será necesario desarrollar un Plan de Garantía de Calidad, de acuerdo con el estándar elegido y teniendo en cuenta los aspectos específicos del proyecto que van a influir en la calidad. El Plan de Garantía de Calidad va a describir en detalle que actividades de garantía de calidad aplicar en cada una de las tases del desarrollo.

Finalmente, los informes de calidad contendrán los resultados obtenidos con la aplicación de las diferentes actividades de garantía de calidad.

4.2. El Manual de Calidad

El Manual de Calidad:

- Formaliza la política de la empresa relativa a la gestión de calidad.
- Define los requisitos generales que deben ser establecidos en la empresa para garantizar la implantación del sistema de calidad y su cumplimiento.
- Detalla los criterios a seguir y hace referencia a los procedimientos que componen el sistema de calidad de la empresa.

La estructura del Manual de Calidad, según el estándar ISO 9004-2, debe tener los siguientes apartados:

1. Generalidades.
2. Organización.
3. Revisión del contrato.
4. Control del proyecto.
5. Control de la documentación.
6. Control de las compras.
7. Productos o servicios suministrados por el cliente.
8. Identificación y trazabilidad.
9. Control de los procesos.
10. Inspección y ensayos.
11. Verificación de los equipos de inspección.
12. Estados de inspección.
13. Control de productos o servicios no conformes.
14. Acciones correctivas.
15. Manipulación, almacenamiento, embalado y entrega.
16. Registros sobre la calidad.
17. Auditorias internas.
18. Formación y adiestramiento.
19. Mantenimiento del producto o servicio.
20. Técnicas estadísticas.

4.3. Guía de IEEE para la planificación de la Garantía de Calidad

Esta guía viene a complementar el estándar de IEEE para los Planes de Garantía de Calidad y recoge el consenso alcanzado entre un cierto número de personas experimentadas en la generación, implementación, evaluación y modificación de Planes de Garantía de Calidad de software sobre lo que son buenas prácticas de garantía de calidad. Es, por tanto, un conjunto de recomendaciones, no un estándar.

4.3.1. Tipo de software considerado

El estándar de IEEE está dirigido al desarrollo de software crítico, es decir, aquel cuyo fallo puede producir grandes pérdidas o catástrofes. Si se está desarrollando software no crítico, no tiene sentido imponer todos los requisitos del estándar.

4.3.2. Grupos afectados por la garantía de calidad

- Los usuarios, ya sean externos o internos a la organización que desarrolla el software. Lo que necesita el usuario es que, al finalizar el proceso de desarrollo, el producto satisfaga los requisitos que se han identificado. El programa de garantía de calidad permite al usuario tener un cierto grado de confianza, durante el mismo

proceso de desarrollo, en que el producto está adquiriendo dichos requisitos, sin tener que esperar a la realización de pruebas al final izar el proceso de desarrollo para confirmar que los posee o averiguar que no los posee.

- Los desarrolladores. El programa de garantía de calidad les ofrece un marco de trabajo estándar y estable de un proyecto a otro, sobre el que basar las responsabilidades.
- El público en general, ya que se verá afectado por el buen o mal funcionamiento de los sistemas software desarrollados.

4.3.3. ¿Para qué puede ayudar la guía?

- Para desarrollar o implementar un Plan de Garantía de Calidad para un proyecto. Esta tarea es propia del personal de garantía de calidad.
- Para iniciar la puesta en práctica de procedimientos de garantía de calidad en un proyecto. Esta tarea es propia de los Jefes de Proyecto de Desarrollo de software.
- Para evaluar o especificar un Plan de Garantía de Calidad. Esta tarea es propia de compradores o usuarios.

4.3.4. Estructura de un Plan de Garantía de Calidad

El estándar impone la inclusión de 13 secciones en el Plan, y en el orden en que aparecen a continuación, y admite la inclusión de secciones adicionales al final del documento si es necesario.

1. Propósito

Aquí se delimita el propósito y alcance del Plan. Deben listarse los nombres de todos los productos software a los que afecta el Plan, e indicarse el propósito del software.

Al rellenar esta sección, deberían darse respuesta a las siguientes cuestiones:

- ¿Cuáles son los productos software cubiertos por este Plan?
- ¿Para qué se va a utilizar el software cubierto por este Plan? ¿Es un software crítico? ¿Es parte de un sistema más grande?, y en este caso ¿de qué forma se relaciona con dicho sistema?
- ¿Por qué se está escribiendo este Plan? ¿Responde a un requisito externo o interno? ¿Por qué es este Plan necesario?
- ¿En qué documentos (estándares o guías) se basa este Plan? ¿En qué medida?
- ¿Cómo se justifican las desviaciones sobre los documentos mencionados? ¿Cuáles son los atributos del producto o del desarrollo que justifican la imposición de prácticas o procedimientos adicionales o bien la eliminación o suavización de los mismos?

2. Documentos de referencia

Aquí se proporciona una lista completa de todos los documentos referenciados en el Plan.

3. Gestión

Aquí se describe la organización, tareas y responsabilidades.

a) Organización.

Aquí se describe la estructura organizativa, cada uno de los elementos principales de la organización, y la delegación de responsabilidades, en la medida en que esto influye en la calidad del software.

Se deben describir claramente las dependencias organizativas y funcionales entre aquellos elementos responsables de la garantía de calidad y los elementos responsables del desarrollo o uso.

Los elementos organizativos responsables de las funciones de garantía de calidad pueden ser:

- Desarrolladores con un conocimiento especial de las técnicas y herramientas de garantía de calidad.
- Un elemento dedicado a garantía de calidad que dé servicio a varios proyectos.
- Un conjunto de elementos organizativos separados, cada uno de los cuales implementa ciertas funciones de garantía de calidad.

En el estándar no se impone ningún tipo de estructura organizativa.

En cualquier caso, se deben describir las interrelaciones entre los elementos responsables de las funciones de garantía de calidad y otros elementos.

Se debería incluir una representación gráfica de la estructura organizativa y una explicación textual que debería incluir:

- Una descripción de cada elemento que interactúe con elementos de garantía de calidad.
- Responsabilidades delegadas en cada uno de los elementos que interactúan.
- A quién o quiénes informan cada uno de los elementos que interactúan.
- Identificación de aquel elemento organizativo con autoridad en la liberación de versiones de productos.
- Identificación de aquel elemento organizativo que aprobará el Plan.
- El método que se utilizará para resolver conflictos entre los elementos.

También podría incluir:

- El tamaño de cada elemento de garantía de calidad.
- Una explicación de posibles desviaciones respecto de las políticas, procedimientos o estándares de garantía de calidad de la organización.

b) Tareas.

Aquí se describen aquellas tareas asociadas con la porción del ciclo de vida que cubre el Plan, poniendo más énfasis en las actividades de garantía de calidad. También se indicará la secuencia de dichas tareas.

Las tareas básicas que se incluirán en este punto son las que se describen en las secciones 4 a 13 del Plan, si son aplicables. Se debe explicar cualquier omisión o desviación respecto de alguna de las tareas especificadas en el estándar. También se pueden incluir tareas adicionales, correspondientes a las secciones adicionales del Plan.

Para cada tarea se debe describir sus criterios de entrada y de salida, es decir, qué se necesita para iniciar la tarea y cuáles son sus salidas. Las salidas se definirán de tal forma que sea posible determinar de una forma objetiva, según un procedimiento establecido, si la salida se ha completado o no.

c) Responsabilidades.

Aquí se identifican los elementos organizativos responsables de cada tarea.

En esta sección se designa también al personal responsable de la publicación, distribución, mantenimiento e implementación del Plan de Garantía de Calidad.

4. Documentación

Aquí se identifica toda la documentación que gobernará el desarrollo, validación y verificación, mantenimiento y uso del software.

También se describirá la revisión o auditoria que se utilizará para determinar la adecuación de cada uno de estos documentos.

La documentación mínima que exige el estándar para garantizar que la implementación del software satisface los requisitos es la siguiente:

- Especificación de requisitos software.
- Descripción del diseño del software.
- Plan de verificación y validación.
- Informe de verificación y validación.
- Documentación de usuario.

5. Estándares, prácticas y convenciones

Aquí se identifican todos los estándares, practicas y convenciones que se van a aplicar.

También se describirá la forma en que se va a monitorizar y asegurar el cumplimiento de estos elementos.

6. Revisiones y auditorias

En esta sección se definen todas las revisiones y auditorias técnicas y de gestión que se llevarán a cabo.

También se describirá la forma en que dichas revisiones y auditorias se llevarán a cabo.

Se deben llevar a cabo, al menos, las siguientes revisiones y auditorias:

- Revisión de los requisitos software.
- Revisión del diseño preliminar.
- Revisión del diseño crítico.
- Revisión del plan de verificación y validación.
- Auditoría funcional.
- Auditoría física.
- Auditorías del proceso.
- Revisiones de gestión.

Se podrían planificar otras revisiones como, por ejemplo, la revisión de la documentación de usuario.

Para cada tipo de revisión, se debe explicar:

- Su objetivo.
- Qué producto es el que se evalúa.
- Sus propósitos.
- Cuál es el elemento organizativo responsable de llevar a cabo la revisión.
- Cuáles son los elementos organizativos que deben tomar parte en la revisión.

- Cuáles son los requisitos de revisión.
- Dónde deben documentarse los resultados de la revisión.

Para cada tipo de auditoría se debe explicar:

- Su objetivo.
- Cuál es el elemento organizativo responsable de llevar a cabo la auditoría.
- Donde de deben documentarse los resultados de la auditoría.
- Cuáles son las entradas para la auditoría.

Revisión de los requisitos software: su objetivo es asegurar la adecuación, factibilidad técnica y completitud de los requisitos incluidos en la especificación de requisitos software. Debe evaluar la especificación de requisitos software para ver si cumple los atributos que detalla el estándar IEEE 830-1984.

Revisión del diseño preliminar: su objetivo es asegurar la adecuación del diseño preliminar tal y como aparece en una versión preliminar de la descripción de diseño del software, antes de comenzar con el diseño detallado.

Revisión del diseño crítico: su objetivo es asegurar la adecuación del diseño detallado tal y como aparece en la versión final de la descripción de diseño del software, antes de comenzar con la codificación.

Revisión del Plan de Verificación y Validación: su objetivo es evaluar adecuación y lo completo de los métodos de verificación y validación definidos en el Plan de Verificación y Validación.

AuditorÍa funcional: esta auditoría se lleva a cabo antes de la entrega del software para comprobar que se han satisfecho todos los requisitos especificados en la especificación de requisitos de software.

Auditoría física: esta auditoría se lleva a cabo para comprobar que el software y su documentación son consistentes internamente y están listos para su entrega. Para ello se compara el código con su documentación de apoyo.

Auditorías del proceso: son auditorías en las que se examinan muestras de los diferentes productos del desarrollo para comprobar la consistencia del producto según evoluciona a través del proceso de desarrollo. Con ello se obtienen medidas de lo bien que funciona el proceso.

Revisiones de gestión: estas auditrías se llevan a cabo periódicamente para valorar la ejecución de este Plan.

7. Gestión de configuración

Aquí se describen los métodos que se vana utilizar para llevar acabo las diferentes tareas de gestión de configuración. Puede ser una referencia al Plan de Gestión de Configuración, si éste existe.

8. Gestión de problemas y acciones correctivas

Aquí se describen las prácticas y procedimientos que se van a utilizar para la notificación, seguimiento y resolución de problemas software, así como las responsabilidades organizativas.

El propósito de un sistema de gestión de problemas y acciones correctivas es:

- Asegurar que todos los problemas se documentan, se corrigen y no caen en el olvido.
- Asegurar que se evalúa la validez de los informes de problemas.
- Realimentar al desarrollador y al usuario sobre el estado de los problemas.
- Proporcionar datos para medir y predecir la calidad y fiabilidad del software.

9. Herramientas, técnicas y metodologías.

En esta sección se identifican todas las herramientas, técnicas y metodologías que se van a utilizar en el desarrollo que apoyan la garantía de calidad, se hace constar su propósito y se describe su uso.

Algunas de las herramientas que pueden incluirse son:

- Utilidades del sistema operativo.
- Depuradores.
- Ayudas para documentación.
- Preprocesadores.
- Comparadores de ficheros.
- Analizadores de estructura.
- Monitores de rendimiento.
- Paquetes de análisis estadístico.
- Generadores de casos de prueba.
- Ejecutores de pruebas.
- Herramientas de control estático o dinámico.

Algunas de las técnicas que pueden ayudar a la evaluación o mejora de la calidad son:

- Estándares.
- Inspecciones.
- Verificación de requisitos y diseño.
- Traza de requisitos.
- Valoraciones y medidas de fiabilidad.
- Análisis lógicos formales o rigurosos.

Las metodologías de garantía de calidad serán conjuntos integrados de técnicas, de entre las anteriores.

10. Control del código

En esta sección se definen los métodos y facilidades que se van a utilizar para controlar el almacenamiento y mantenimiento de versiones del código.

Aquí se debería especificar un procedimiento de control del código que:

- Defina cuál es el software que se va a controlar.
- Describa un método estándar para identificar, etiquetar y catalogar el software.
- Liste la localización física del software bajo control.
- Describa la localización, forma de mantenimiento y de uso de las copias de seguridad.
- Describa los procedimientos para distribución de copias.
- Identifique la documentación que se verá afectada por los cambios.
- Describa los procedimientos para la construcción de una nueva versión.

11. Control de medios

En esta sección se definen los métodos y facilidades que se van a utilizar para proteger el medio físico de accesos no autorizados y daños y degradaciones inesperadas.

Se debería asegurar que:

- Está garantizado el almacenamiento y recuperación de software.
- El software está accesible únicamente para aquellos que lo necesitan.
- Se controla el entorno para que no se degrade el medio físico en el que se almacena el software.
- Se almacenan copias del software crítico y del código en línea base fuera de las instalaciones de la organización.

12. Control de suministradores y subcontratas

En esta sección se explica de qué forma se va a asegurar que el software comprado o subcontratado cumple los requisitos técnicos.

13. Recolección, mantenimiento y retención de registros.

En esta sección se identifica aquella documentación que se debe retener, y se especifican los métodos y facilidades que se utilizarán para recolectar, proteger y mantener esta documentación. También se especificará el período de retención para cada tipo de registro.

Se puede registrar no sólo documentación, sino también los medios físicos que contienen las versiones de los programas y los materiales utilizados en las pruebas, para asegurar la repetibilidad de los tests en el futuro.

4.4. ¿Cómo implementar un Plan de Garantía de Calidad?

Hay varios aspectos imprescindibles para implementar con éxito un Plan de Garantía de Calidad para un proyecto específico:

1. Que sea aceptado por la dirección.

   Esta aceptación debe incluir el compromiso de facilitar los recursos necesarios para implementar las actividades de garantía de calidad.

   Debe ser aceptado por cada unidad organizativa para la que se hayan definido responsabilidades dentro del Plan de Garantía de Calidad. Esta aceptación refleja con la firma de la persona responsable de dicha unidad en la cubierta del Plan de Garantía de Calidad.

2. Que sea aceptado por el personal de desarrollo.

3. Que se planifique la implementación del Plan de Garantía de Calidad.

   - Identificando los recursos necesarios: personal, equipamiento, facilidades y herramientas.
   - Programando su implementación.
   - Valorando los riesgos.

4. Entrenamiento del personal encargado de implementar el Plan de Garantía de Calidad.

5. Distribución adecuada del Plan de Garantía de Calidad.

6. Ejecución del Plan de Garantía de Calidad.

Una vez distribuido, los elementos de garantía de calidad deben asegurarse de que todas las tareas documentadas en el Plan de Garantía de Calidad se ejecutan correctamente. Los documentos de trabajo asociados a las revisiones y auditorías serán evidencia suficiente de que todos los pasos del Plan de Garantía de Calidad se han ejecutado y revisado.

4.5. El Plan General de Garantía de Calidad

Elaborado por el Ministerio para las Administraciones Públicas para el Consejo Superior de Informática en 1991. Debe ser aplicado en todos los proyectos de desarrollo de software que se llevan a cabo en la Administración española.

Tiene una estructura con cuatro componentes:

1. Guía metodológica para la elaboración de Planes de Garantía de Calidad.
2. Esquema formal para la clasificación dé proyectos informáticos.
3. Procedimientos de control de calidad.
4. Instrumentos de control y elementos auxiliares de control de calidad.

A continuación vamos a ver brevemente en qué consiste cada uno de ellos.

4.5.1. La guía metodológica

En primer lugar, define los agentes que deben intervenir en un proyecto informático:

USR (Usuario o cliente)

Es el que demanda el desarrollo y, por lo tanto, su calidad interviene en:

- ✓ Ciertas revisiones
- ✓ Pruebas de aceptación

DIR (Director del proyecto)

Es el responsable último frente al usuario. Se encarga de aprobar y de impulsar la ejecución y puesta en práctica del Plan de Garantía de Calidad específico del proyecto.

EDS (Equipo de Desarrollo)

Es el sujeto pasivo de los procedimientos del control de calidad

EGC (Equipo de Garantía de Calidad)

Diseña el Plan de Garantía de Calidad específico. Es sujeto activo de los procedimientos de control de calidad y a su vez es sujeto pasivo de las auditorías externas del sistema de calidad.

AUD (Equipo de Auditoría)

Sólo interviene en procedimientos extraordinarios de auditoría

En proyectos pequeños y medianos algunos de estos roles recaen en las mismas personas, por ejemplo DIR= EGC, y normalmente no existe AUD.

La guía metodológica también incluye una metodología para la elaboración de Planes de Garantía de Calidad específicos. Esta metodología incluye las siguientes fases:

1ª. Actuaciones preliminares.

2ª. Caracterización del proyecto a efectos de garantía de calidad. Para ello se utiliza el segundo componente del Plan General de Garantía de Calidad, es decir, el esquema formal para la clasificación de proyectos informáticos.

3ª. Selección y adaptación de procedimientos de control de calidad. Para ello se utiliza el tercer componente del Plan General de Garantía de Calidad.

4ª. Selección y adaptación de instrumentos de control y elementos auxiliares de garantía de calidad. Para ello se utiliza el cuarto componente del Plan General de Garantía de Calidad.

5ª. Redacción y aprobación del Plan General de Garantía de Calidad específico.

4.5.2. Esquema para la clasificación dé proyectos

Establece una serie de atributos que deben ser valorados para cada proyecto:

DIM: Dimensión.

COM: Complejidad.

FIAB: Requisitos de fiabilidad.

SEC: Requisitos de seguridad.

CEX: Requisitos de comportamiento externo.

CIN: Requisitos de comportamiento interno.

DESP: Grado de definición, estructura y modularidad de las especificaciones.

TPMV: Tipología de la máquina virtual.

FCMV: Funcionalidad de la máquina virtual.

DHMV: Grado de distribución y heterogeneidad de la máquina virtual de implantación.

CRT: Carga de trabajo.

INT: Nivel de interacción con otras aplicaciones o datos.

DIFE: Diferencias entre los entornos de desarrollo y de implantación.

COST: Coste total estimado del proyecto.

PLZ: Plazo estimado de desarrollo.

EPRY: Estabilidad del Proyecto.

ECON: Evaluación previa del contratista.

REC: Disponibilidad de recursos para garantía de calida.

También proporciona una serie de categorías de riesgos que pueden afectar al proyecto:

RI: Defectos graves y recurrentes por mala adecuación funcional, falta de fiabilidad, , problemas de seguridad, etc.

R2: Baja calidad en los productos de las fases de desarrollo.

R3: Dificultades graves de implantación por mala aplicación, al entorno real de implantación.

R4: Imposibilidad de mantener los costes de desarrollo en relación con los límites establecidos en la contratación.

R5: Incumplimiento grave de los plazos de ejecución.

R6: Imposibilidad de gestionar y controlar el desarrollo del proyecto.

R7: Inconclusión del proyecto.

Las tareas que se deben realizar en esta fase de la metodología para la elaboración de un Plan de Garantía de Calidad específico son:

1. Construir el diagrama característico de la aplicación: consiste en dar una valoración entre 1 y 5 para cada uno de los atributos anteriores.
2. Elección del modelo de referencia para el desarrollo:
   - Ciclo de vida.
   - Modelo de proceso.
3. Determinar el perfil de riesgos potenciales: consiste en calcular el coeficiente de divergencia para cada uno de los tipos de riesgos anteriores. El resultado será un valor en el intervalo [- 5,5]. El cálculo se realiza en función del diagrama característico de la aplicación.
4. Determinación del foco de interés en garantía de calidad: dependiendo del tipo de proyecto de que se trate, las actividades de garantía de calidad deberán focalizarse en cosas diferentes. En este paso se determinan las áreas prioritarias.

4.5.3. Procedimientos de control de calidad

En el Plan General de Garantía de Calidad se contemplan los siguientes tipos de actividades de control de calidad:

- Revisiones, que pueden ser:
  - Mínimas.
  - Técnicas formales.
  - Inspecciones detalladas.
- Pruebas que pueden ser:
  - De validación de módulos.
  - De integración de módulos y componentes.
  - De aceptación de la aplicación.
- Procedimientos extraordinarios.
- Auditorías.
- Procedimientos particulares.
- Evaluación de prototipos.

4.5.4. Instrumentos de control y elementos auxiliares de control

Se consideran instrumentos de control los siguientes:

- Listas de comprobación (listas de control).
- Guiones de recomendaciones, para:
  - Revisiones.
  - Pruebas.
  - Auditorías.

Se consideran elementos auxiliares los siguientes:

- Formatos de referencia para los documentos producidos como resultado de las funciones de control de calidad.
- Dossier de garantía de calidad del proyecto.

Índice